9.3　输入对话框

前面在介绍 JavaScript 的时候，曾介绍过它的三个基本对话框 alert()、confirm()、prompt()。它们分别表示提示对话框、确认对话框、输入对话框。从本节开始介绍如何通过 jQuery 来模拟创建这三个对话框。本节创建输入对话框，其实现原理是：利用弹出层，在层上添加用户输入功能，并在层隐藏后将用户输入内容显示在页面上。其中，使用到了 jQuery 函数 ready()、click()、css()、show()、hide()、text()、val()。

其功能实现如下。

（1）为页面中触发弹出对话框的元素添加单击事件，在事件中触发对话框所在层显示，并定位对话框的具体显示位置。

（2）在触发对话框事件中显示遮盖层。

（3）为对话框添加触发隐藏的单击事件，同时隐藏遮盖层。

（4）为对话框的内容提交按钮添加隐藏对话框和遮盖层功能，并在页面中显示用户输入的内容。

首先，对上一节的对话框中的样式进行修改：

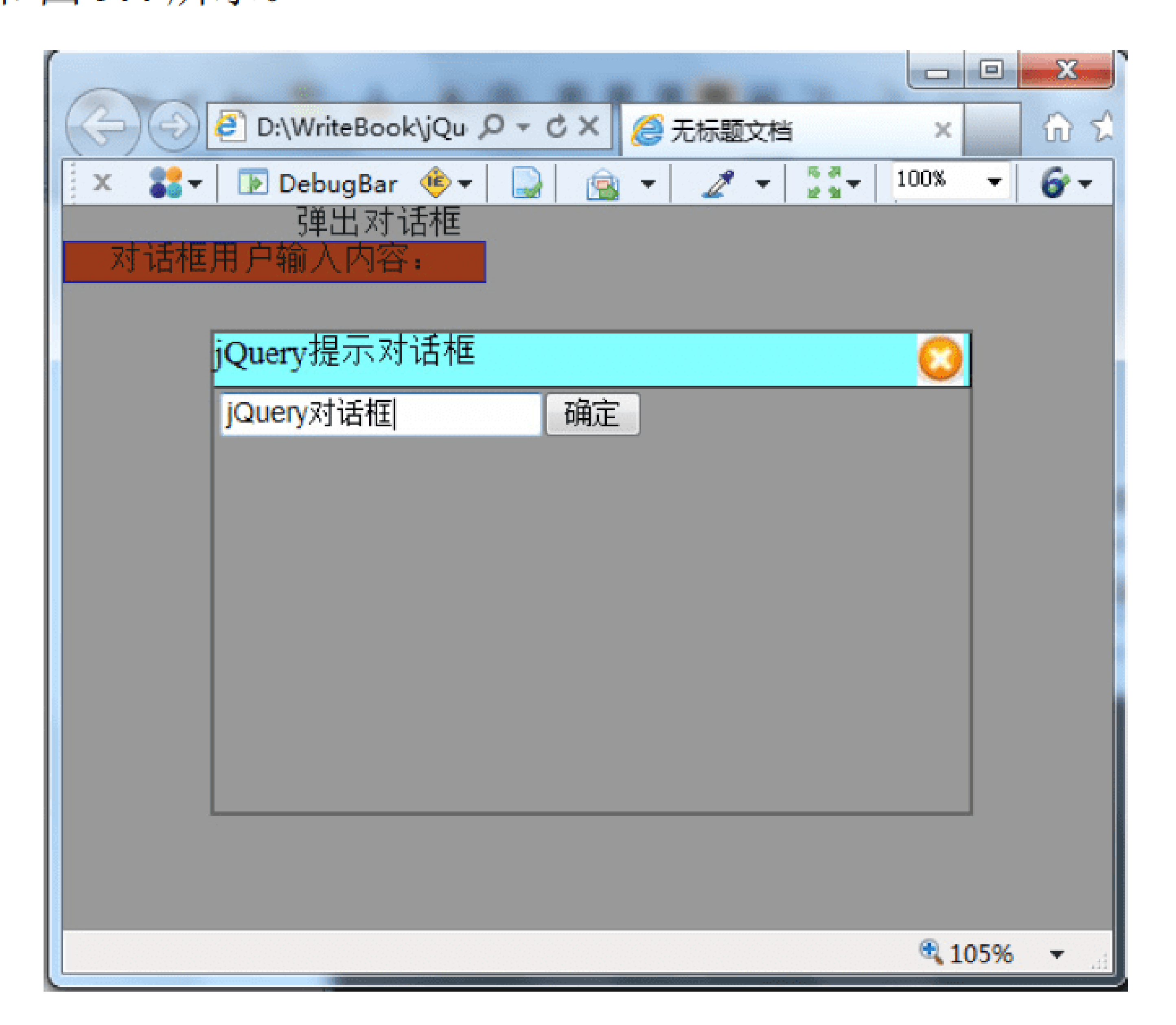
```
1    <div id="btn">弹出对话框</div>
2    <p id="result">对话框用户输入内容：</p>
3    <div id="showMessage">
4        <div id="titlearea">
5            <div id="close"></div>
6        <div id="titles">jQuerys 输入对话框</div>
7        </div>
8        <div id="content"><input id="userdata" /><input type="button"
         id="tijiao" value="确定" /></div>
9    </div>
10   <div id="background"></div>
```

然后，修改 JavaScript 功能代码：

```
1   <script type="text/JavaScript">
2       $(document).ready(function(){
3           $("#btn").click(function(){     //触发输入对话框
            //显示遮盖层
4               $("#background").css("width",$(window).width()+"px")
5                   .css("height",$(window).height()+"px").show();
            //显示输入对话框
6               $("#showMessage").css("left",($(window).width()-$
                ("#showMessage").width())/2+"px")
7               .css("top",($(window).height()-$("#showMessage").
                height())/2+"px").show();});
            //隐藏遮盖层和对话框
8           $("#close").click(function(){
9           $("#background").css("display","none");
10          $("#showMessage").hide();
```

```
11       });
         //隐藏遮盖层和对话框，在页面上显示输入内容
12       $("#tijiao").click(function(){
13           $("#background").css("display","none");
14           $("#showMessage").hide();
15           $("#result").text($("#result").text()+$("#userdata").val());
16       });
17   });
18   </script>
```

上述代码第 12～16 行是对话框中内容提交按钮的单击事件，在这个事件中隐藏遮盖层，隐藏对话框，并将用户在输入对话框中输入的内容在页面中显示出来，效果如图 9.3 和图 9.4 所示。





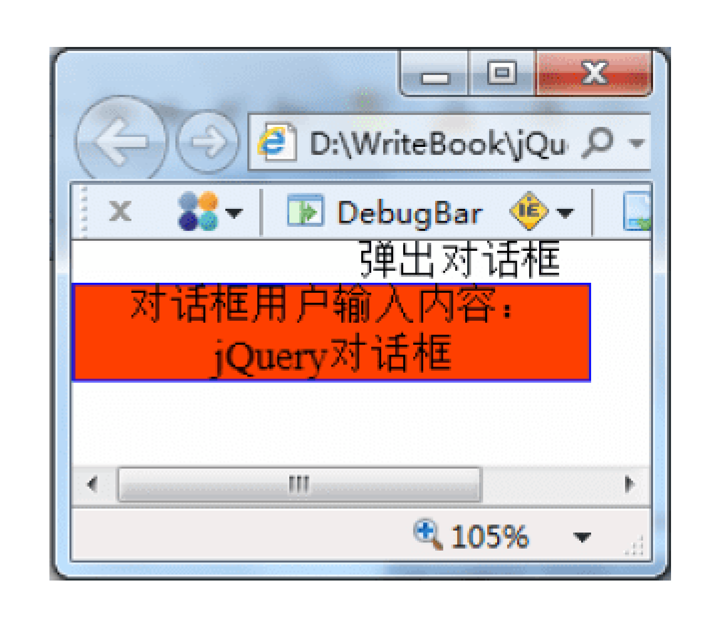


图 9.3　jQuery 输入对话框　　　　图 9.4　jQuery 输入对话框返回内容

9.4　提示对话框

本节将介绍提示对话框。提示对话框主要显示提示信息。它的基本工作原理和 9.2 节所介绍的模式对话框类似，当用户输入相应内容并触发对话框弹出时，自动将用户的内容携带进对话框。其中，要使用到 jQuery 函数 ready()、click()、css()、show()、hide()、attr()。其功能实现如下。

（1）为页面中触发弹出对话框的元素添加单击事件，在事件中触发对话框所在层显示，并定位对话框的具体显示位置，将用户输入内容填入对话框。

（2）在触发对话框事件中显示遮盖层。

（3）为对话框添加触发隐藏的单击事件，同时隐藏遮盖层。

首先，对上一节的对话框中的样式进行修改：

```
1    <div id="btn">弹出对话框<br />
2    <input type="checkbox" id="public" value="0" name="sss" name=
    "public"/>公开资料内容
3    </div>
4    <!--<p id="result">对话框用户输入内容：</p>-->
5    <div id="showMessage">
6        <div id="titlearea">
7            <div id="close"></div>
8        <div id="titles">jQuery 提示对话框</div>
9        </div>
10       <div id="content"><span id="message">您选择了</span><input type=
         "button" id="tijiao" value="确定" /></div>
11    </div>
12   <div id="background"></div>
```

然后，修改 JavaScript 功能代码：

```
1   <script type="text/JavaScript">
2       $(document).ready(function(){
3           var msg=undefined;
4           $("#btn").click(function(){
                    //弹出提示对话框和遮盖层，在提示对话框中显示用户提交内容
5               if($("#public").attr("checked"))
6                   msg="公开资料内容";
7               else
8                   msg="不公开资料内容";
9               $("#background").css("width",$(window).width()+"px")
10                  .css("height",$(window).height()+"px").show();
11              $("#showMessage").css("left",($(window).width()-$
                ("#showMessage").width())/2+"px")
12              .css("top",($(window).height()-$("#showMessage").
                height())/2+"px").show();
13              $("#message").text($("#message").text()+msg);
14          });
15          $("#close").click(function(){        //隐藏对话框和遮盖层
16              $("#background").css("display","none");
17              $("#showMessage").hide();
18          });
19          $("#tijiao").click(function(){       //隐藏对话框和遮盖层
20          $("#background").css("display","none");
21          $("#showMessage").hide();
22          msg="";
23          $("#message").text("您选择了");
24          //$("#result").text($("#result").text()+$("#userdata").
            val());
25      });
26  });
27  </script>
```

上述代码第 5～8 行提取了用户在页面上的选择内容，并记录下来。第 13 行将用户的选择内容填入对话框，提示用户操作。第 22、23 行，当对话框隐藏的时候将其内容进行重

新设置，以便下次使用，效果如图 9.5 和图 9.6 所示。

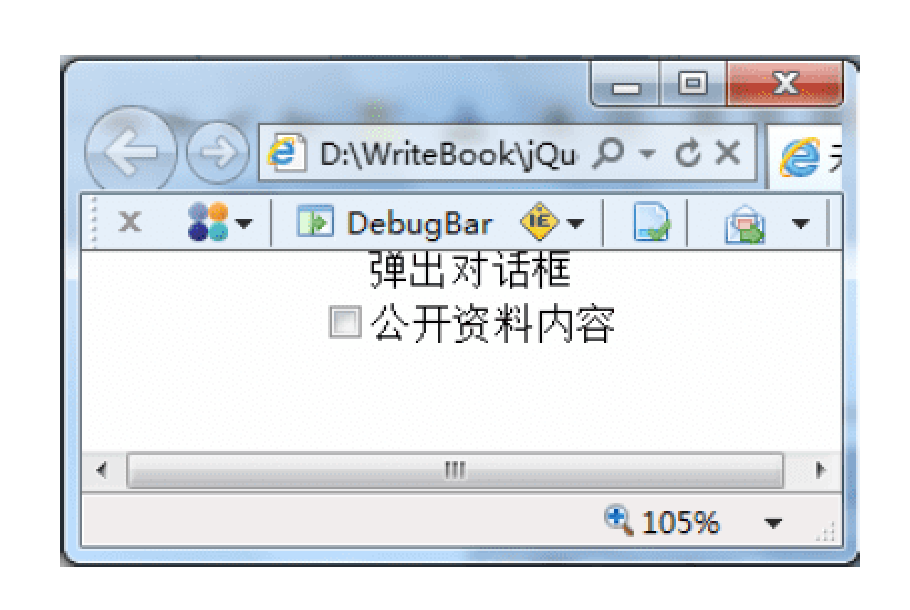


图 9.5　用户输入选择内容

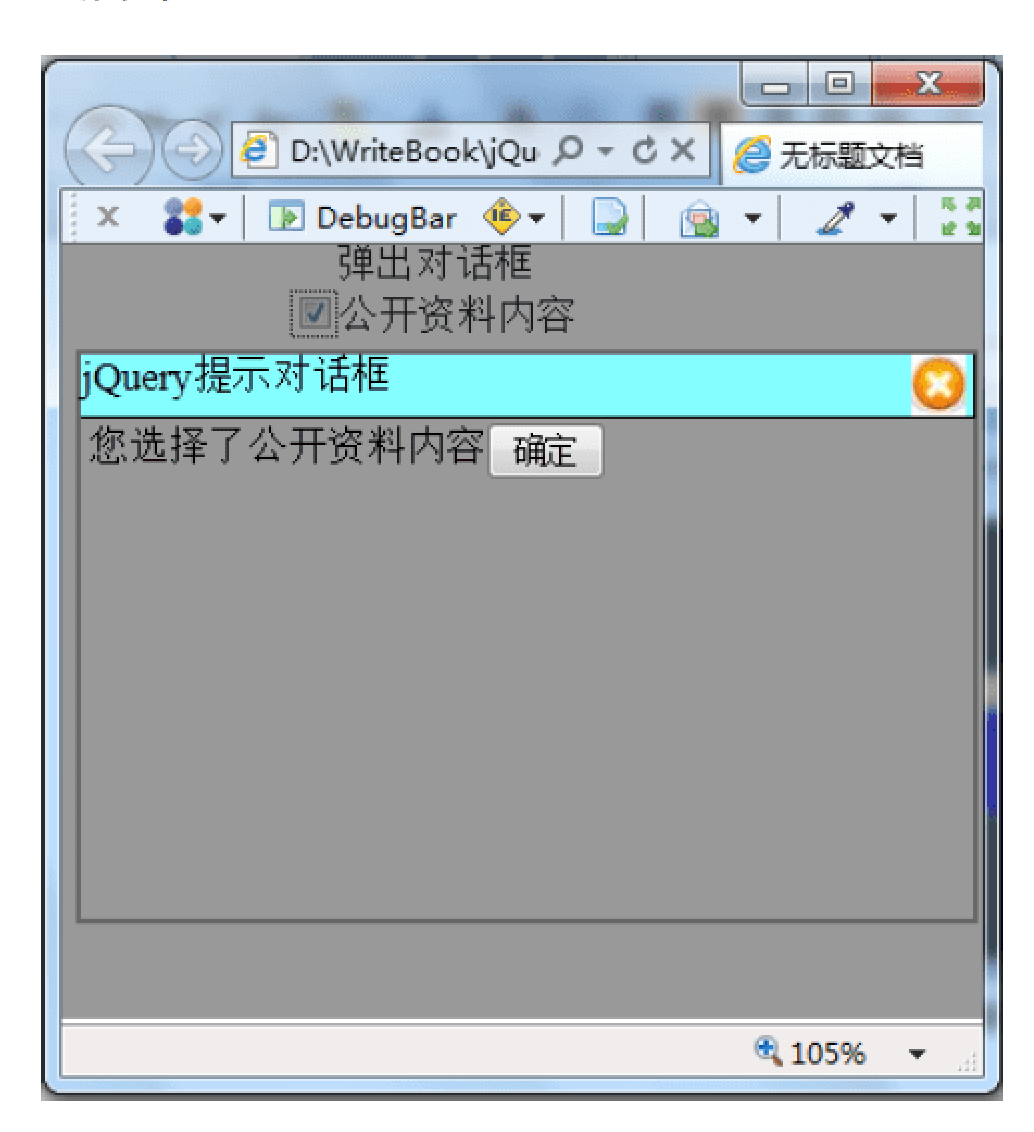


图 9.6　提示对话框

9.5　确认对话框

前面讨论了输入对话框和信息提示对话框。下面来看确认对话框如何实现。确认对话框和前两节介绍的对话框的不同在于，它是一种选择性操作对话框。确认对话框为用户提供两种选择结果，并且用户的选择会直接影响页面上的操作。

它的实现原理是：从页面提取用户操作内容在对话框显示时，提示用户是否确认自己的操作。当用户对对话框做出确认或者否定操作时，又返回来直接修改页面内容。其中，使用到了 jQuery 函数 ready()、click()、css()、show()、hide()、attr()。

其功能实现如下。

（1）为页面中触发弹出对话框的元素添加单击事件，在事件中触发对话框所在层显示，并定位对话框的具体显示位置，将用户输入内容填入对话框。

（2）在触发对话框事件中显示遮盖层。

（3）为对话框的确认按钮添加触发隐藏的单击事件，同时隐藏遮盖层。

（4）为对话框的取消按钮添加触发隐藏的单击事件，同时隐藏遮盖层。如果用户勾选了页面内容，则取消勾选操作。

首先，对上一节的对话框中的样式进行修改：

```
1    <div id="btn">弹出对话框<br />
2    <input type="checkbox" id="public" value="0" name="sss" name=
     "public"/>公开资料内容
3    </div>
4    <!--<p id="result">对话框用户输入内容：</p>-->
5    <div id="showMessage">
6       <div id="titlearea">
```

```
7           <div id="close"></div>
8       <div id="titles">jQuery 提示对话框</div>
9       </div>
10      <div id="content"><span id="message">您选择了</span><input type=
        "button" id="tijiao" value="确定" />
11       <input type="button" id="cancel" value="取消" />
12       </div>
13    </div>
14  <div id="background"></div>
```

然后修改 JavaScript 功能代码：

```
1   <script type="text/JavaScript">
2       $(document).ready(function(){
3           var msg=undefined;
4           /*$("#btn").click(function(){
5               $("#background").css("width",$(window).width()+"px")
6                   .css("height",$(window).height()+"px").show();
7               $("#showMessage").css("left",($(window).width()-$
                ("#showMessage").width())/2+"px")
8               .css("top",($(window).height()-$("#showMessage").
                height())/2+"px").show();}); */
9           $("#btn").click(function(){
                        //弹出提示对话框和遮盖层，在提示对话框中显示用户提交内容
10              if($("#public").attr("checked"))
11                  msg="公开资料内容";
12              else
13                  msg="不公开资料内容";
14              $("#background").css("width",$(window).width()+"px")
15                  .css("height",$(window).height()+"px").show();
16              $("#showMessage").css("left",($(window).width()-$
                ("#showMessage").width())/2+"px")
17              .css("top",($(window).height()-$("#showMessage").
                height())/2+"px").show();
18              $("#message").text($("#message").text()+msg);
19          });
20          $("#close").click(function(){       //隐藏对话框和遮盖层
21              $("#background").css("display","none");
22              $("#showMessage").hide();
23          });
24          $("#tijiao").click(function(){
                                    //隐藏对话框和遮盖层，并确定用户选择
25              $("#background").css("display","none");
26              $("#showMessage").hide();
27              msg="";
28              $("#message").text("您选择了");
29              //$("#result").text($("#result").text()+$("#userdata").
                val());
30          });
31          $("#cancel").click(function(){//隐藏对话框和遮盖层，并取消用户选择
32              $("#background").css("display","none");
33              $("#showMessage").hide();
```

```
34                  msg="";
35                  $("#message").text("您选择了");
36                  if($("#public").attr("checked"))
37                      $("#public").attr('checked',false);
38              });
39          });
40      </script>
```

上述代码的基本内容和 9.4 节相同，第 31～38 行是确认对话框的取消按钮功能，第 36、37 行判断用户是否勾选了相应内容，如果勾选了，则取消勾选。效果如图 9.7 所示。

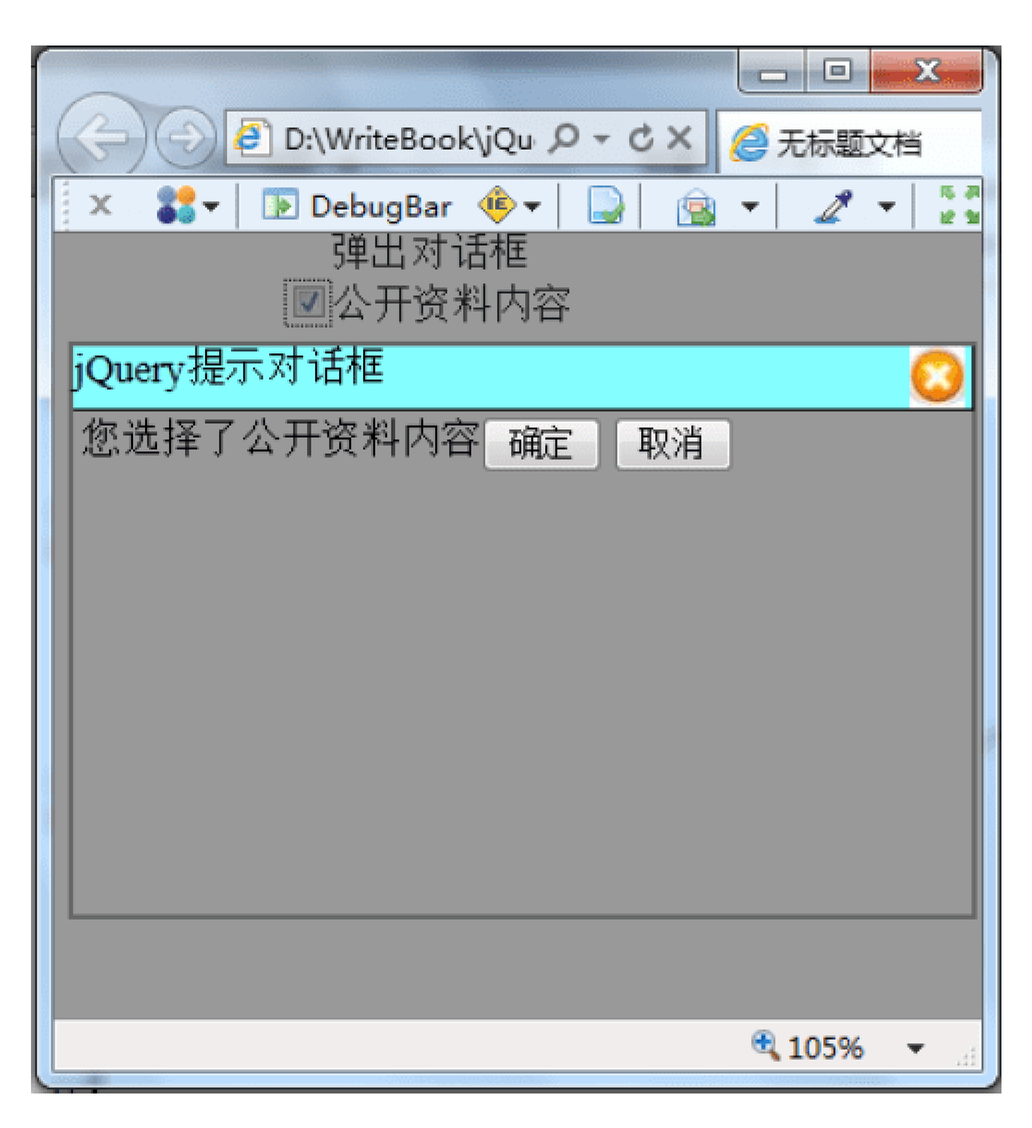


图 9.7　确认对话框

从图 9.7 中可以看到，当勾选复选框后弹出确认对话框。但是，如果尝试单击“取消”按钮，对话框会消失，同时复选框又处于未选中状态。如果单击“确定”按钮，则只是对话框消失，并不影响复选框。

9.6　对话框插件

对话框插件是一种在网页页面上模拟跳出模式或者非模式对话框的插件。它可以帮助我们完成更多的用户交互操作，并节省开发自定义对话框的时间。本节介绍两个对话框插件。它们的基本功能和前面几节介绍的自定义对话框类似。

9.6.1　jqModal 对话框插件

jqModal 对话框插件帮助我们在浏览器中显示告示、对话框、模式窗口。它富有弹性，小巧，就如同“瑞士军刀”一样。它的英文介绍网址为 http://dev.iceburg.net/jquery/jqModal/。

jqModal 特点如下。

（1）设计友好，不用复杂的操作。

（2）翻译友好，不用硬编码英文字符串。

（3）开发友好，利用回调函数实现多功能。

（4）支持远程加载内容。

（5）多模块支持。

jqModal 基本使用步骤如下。

（1）添加对话框容器。

（2）初始化对话框。

（3）触发对话框。

jqModal 中使用的函数如表 9.1 所示。

表 9.1　jqModal函数说明

函　　数	例　　子	说　　明
jqm	$('#dialog').jqm(); $('.dialogs').jqm({ajax:'@href',modal:true});	jqModal 对话框的初始化函数，接收一组参数对象
jqmShow	$('.dialogs').jqmShow();	显示对话框
jqmHide	$('.dialogs').jqmHide();	隐藏对话框
jqmAddTrigger	$('.dialogs').jqmAddTrigger($('#openDialogs a'));	单击触发对话框的页面元素，可以为字符串形式的 jQuery 选择器，也可以为 jQuery 对象，或者一个 DOM 元素
jqmAddClose	$('.dialogs').jqmAddClose($('#hideDialogs a'));	单击关闭对话框的页面元素，可以为字符串形式的 jQuery 选择器，也可以为 jQuery 对象，或者一个 DOM 元素

jqModal 中使用到的参数说明如表 9.2 所示。

表 9.2　jqModal的参数说明

参　　数	数 据 类 型	默　认　值	说　　明
overlay	整型	50	覆盖层的透明度，按照百分比取值
overlayClass	字符串	'jqmOverlay'	覆盖层的 CSS 样式类
closeClass	字符串或者为假	'jqmClose'	单击关闭对话框元素的 CSS 样式类
trigger	字符串、对象、或者为假	'.jqModal'	单击触发对话框的页面元素，可以为字符串形式的 jQuery 选择器，或者一个 DOM 元素，或者不使用触发元素
ajax	字符串或者假	false	指定需要加载远程内容的地址
ajaxText	字符串	''	Ajax 文本内容
target	字符串或者假	false	Ajax 内容加载目标元素，可以为字符串形式的 jQuery 选择器，或者一个 DOM 元素，或者 Ajax 将重写整个对话框内容
modal	布尔型	false	模式对话框参数
toTop	布尔型	false	将对话框置于所有其他元素的上层

下面通过几个示例来了解这个插件的使用方法，这里不涉及 Ajax 内容。

1．默认对话框形式

这种对话框的使用完全套用了所有属性为默认值的情况。首先，需要建立静态页面：

```
1    <a href="#" class="jqModal">view</a>
2    ...
3    <div class="jqmWindow" id="dialog">
4    <a href="#" class="jqmClose">Close</a>
5    <hr>
6    <em>READ ME</em> -->
7    This is a "vanilla plain" jqModal window. Behavior and appeareance extend
     far beyond this.
8    The demonstrations on this page will show off a few possibilites.
     I recommend walking
9    through each one to get an understanding of jqModal <em>before</em> using it.
10    <br /><br />
11    You can view the sourcecode of examples by clicking the JavaScript,
      CSS, and HTML tabs.
12    Be sure to checkout the <a href="README">documentation</a> too!
13    <br /><br />
14    <em>NOTE</em>; You can close windows by clicking the tinted background
      known as the "overlay".
15    Clicking the overlay will have no effect if the "modal" parameter is
      passed, or if the
16    overlay is disabled.
17    </div>
```

上面的 HTML 代码第 1、2 行是触发对话框的页面元素；第 3～17 行是对话框的内容部分。其中“jqmWindow”为对话框的容器层，第 4 行是对话框的关闭元素。然后在<head>中加入 CSS 样式设定：

```
1   <style type="text/css">
2    .jqmWindow {
3       display: none;
4       position: fixed;
5       top: 17%;
6       left: 50%;
7       margin-left: -300px;
8       width: 600px;
9       background-color: #EEE;
10       color: #333;
11       border: 1px solid black;
12       padding: 12px;
13    }
14    .jqmOverlay { background-color: #000; }
15    * html .jqmWindow {
16        position: absolute;
17               top: expression((document.documentElement.scrollTop ||
                document.body.scrollTop) + Math.round(17 * (document.
                documentElement.offse-tHeight || document.body.
                clientHeight) / 100) + 'px');
```

```
18    }
19    </style>
```

第 2～13 行，第 15～18 行是对话框样式的设定，第 14 行是覆盖层的样式。最后引入 jQuery 库文件及 jqModal 插件文件，并加入 JavaScript 代码：

```
1    <script type="text/JavaScript" src="../../jslib/jquery.js"></script>
2    <script type="text/JavaScript" src="JS/jqModal.js"></script>
3    <script type="text/JavaScript">
4        $().ready(function() {
5          $('#dialog').jqm();
6        });
7    </script>
```

代码相对简单，没有任何参数设定，效果如图 9.8 所示。

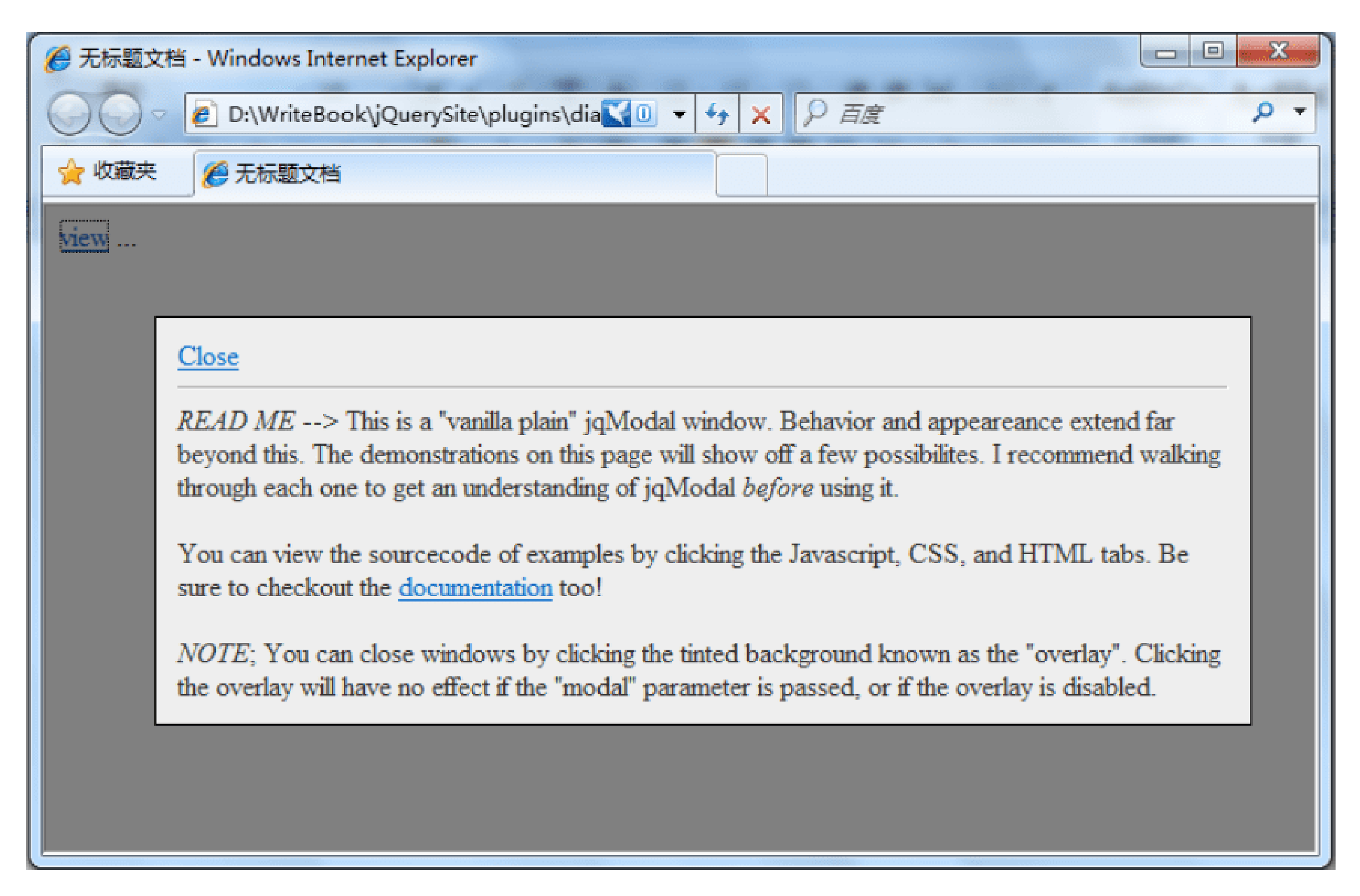


图 9.8　jqModal 默认对话框

2. 可拖动的带特殊背景的模式对话框

这种对话框使用了 jqModal 的一部分属性，并加入了对话框的拖动功能。

首先，建立静态页面。静态页面及 CSS 样式与上面的例子基本相同，只是加入了若干样式设定，具体内容请参考光盘内容。下面看一下具体 JavaScript 功能代码。

```
1    <script type="text/JavaScript">
2        $().ready(function() {
3          $('#ex3a').jqm({
4            trigger: '#ex3aTrigger',
5            overlay: 30, /* 0-100 (int) : 0 is off/transparent, 100 is opaque */
6            overlayClass: 'whiteOverlay'})
7            .jqDrag('.jqDrag'); /* make dialog draggable, assign handle
             to title */
8          $('input.jqmdX')
9          .hover(
```

```
10            function(){ $(this).addClass('jqmdXFocus'); },
11            function(){ $(this).removeClass('jqmdXFocus'); })
12          .focus(
13            function(){ this.hideFocus=true; $(this).addClass
              ('jqmdXFocus'); })
14          .blur(
15            function(){ $(this).removeClass('jqmdXFocus'); });
16        });
17     </script>
```

上述代码中第 3～7 行是 jqModal 插件的基本参数配置，第 4 行表示设定触发元素，第 5 行设定覆盖层的透明度，第 6 行是设定覆盖层的样式类，第 7 行设定对话框拖动时的样式；第 8～15 行是对话框关闭按钮的各种事件的样式设定，包括鼠标悬停、得到焦点、失去焦点，效果如图 9.9 和图 9.10 所示。

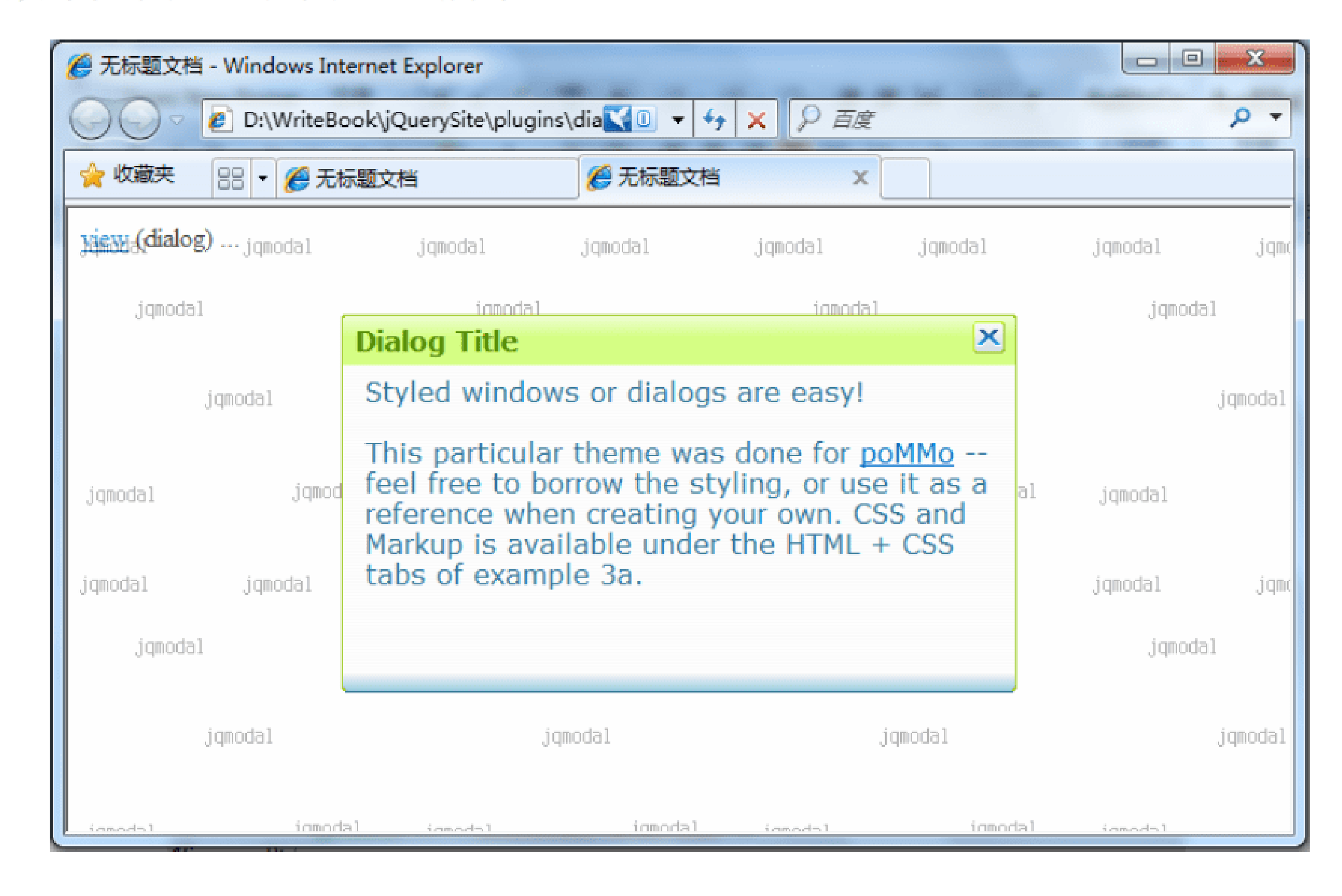


图 9.9　可拖动对话框

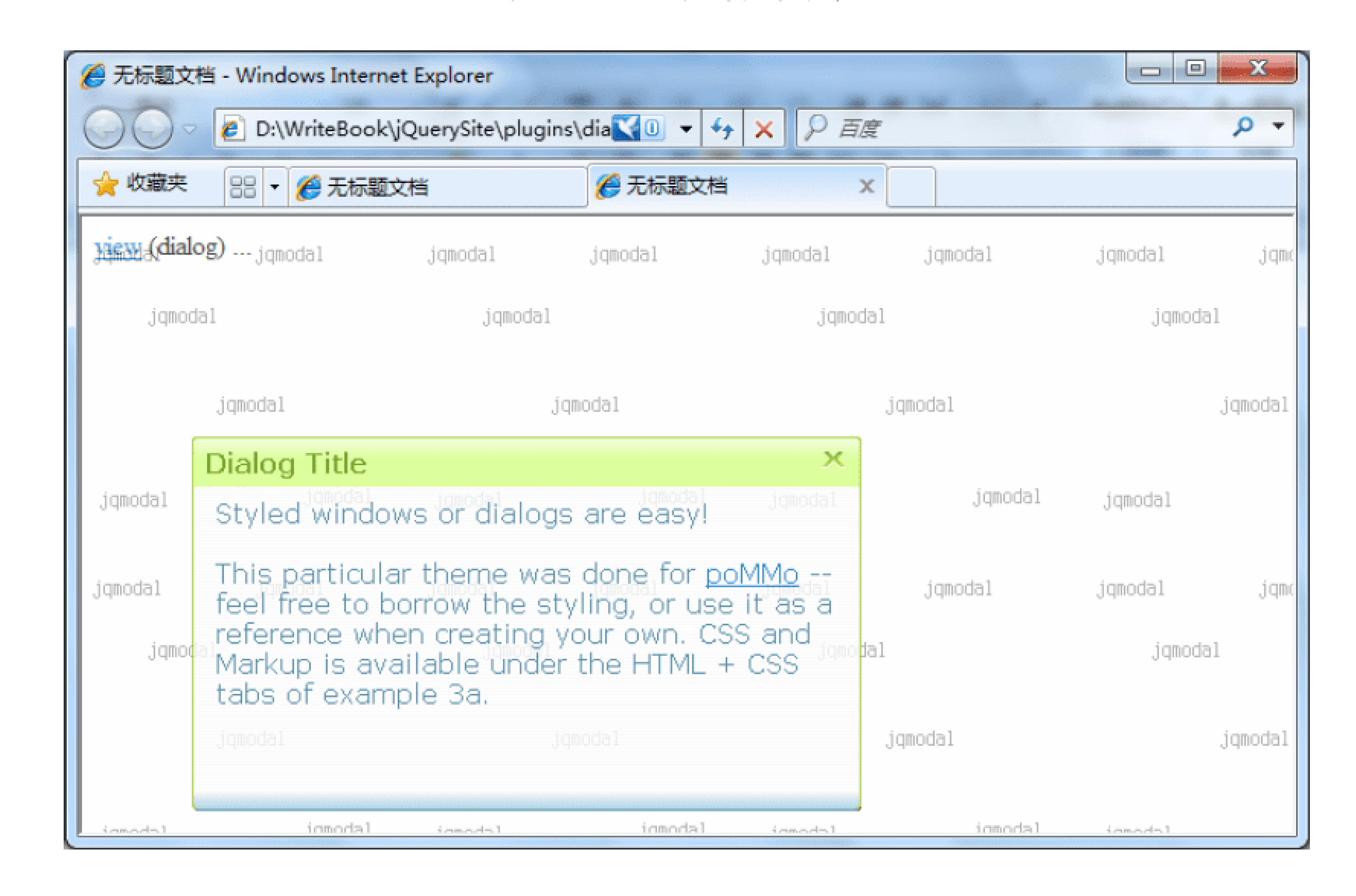


图 9.10　对话框拖动效果

3. jqModal的显示与隐藏

这个例子使用了插件的显示与隐藏方法控制弹出对话框层，同时使用了添加触发对话框的元素和关闭对话框的元素的方法。静态页面和 CSS 代码请参考光盘内容。下面主要讨论一下 JavaScript 的功能代码。

```
1   $().ready(function() {
2     var show = $('#ex5show');
3     var hide = $('#ex5hide');
4     $('div.square')
5       .jqm({overlay: 0, trigger: false})
6       .jqDrag()                  //可拖动操作
7       .jqmAddTrigger(show)
8       .jqmAddClose(hide)
9       $('#ex5multi').click(function() {
10      $('div.square:even').jqmShow();
11      $('div.square:odd').jqmHide();
12      return false;
13    });
14  });
```

上述代码中第 4、5 行进行插件初始化属性设定，覆盖层透明度为 0，不使用默认触发功能。第 7 行添加触发对话框的页面元素。第 8 行添加触发关闭对话框的页面元素。第 9～13 行设定了弹出层的显示效果，奇数层隐藏，偶数层显示。效果如图 9.11 和图 9.12 所示。

这个插件的功能较多，这里挑选了上面三种基本使用方式，其他功能可以参考光盘内容或者去插件的官网查看范例。

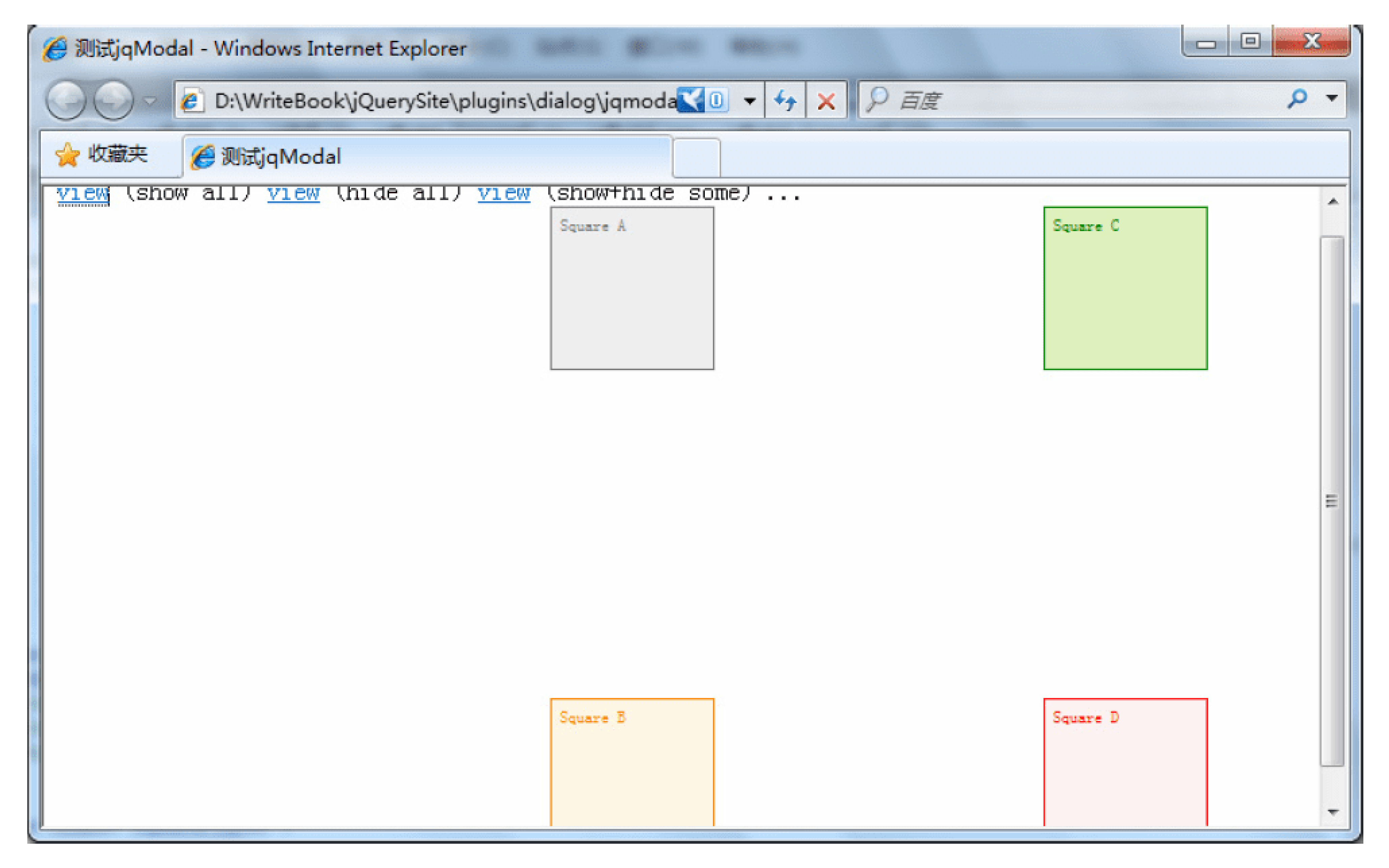


图 9.11　触发显示所有弹出层

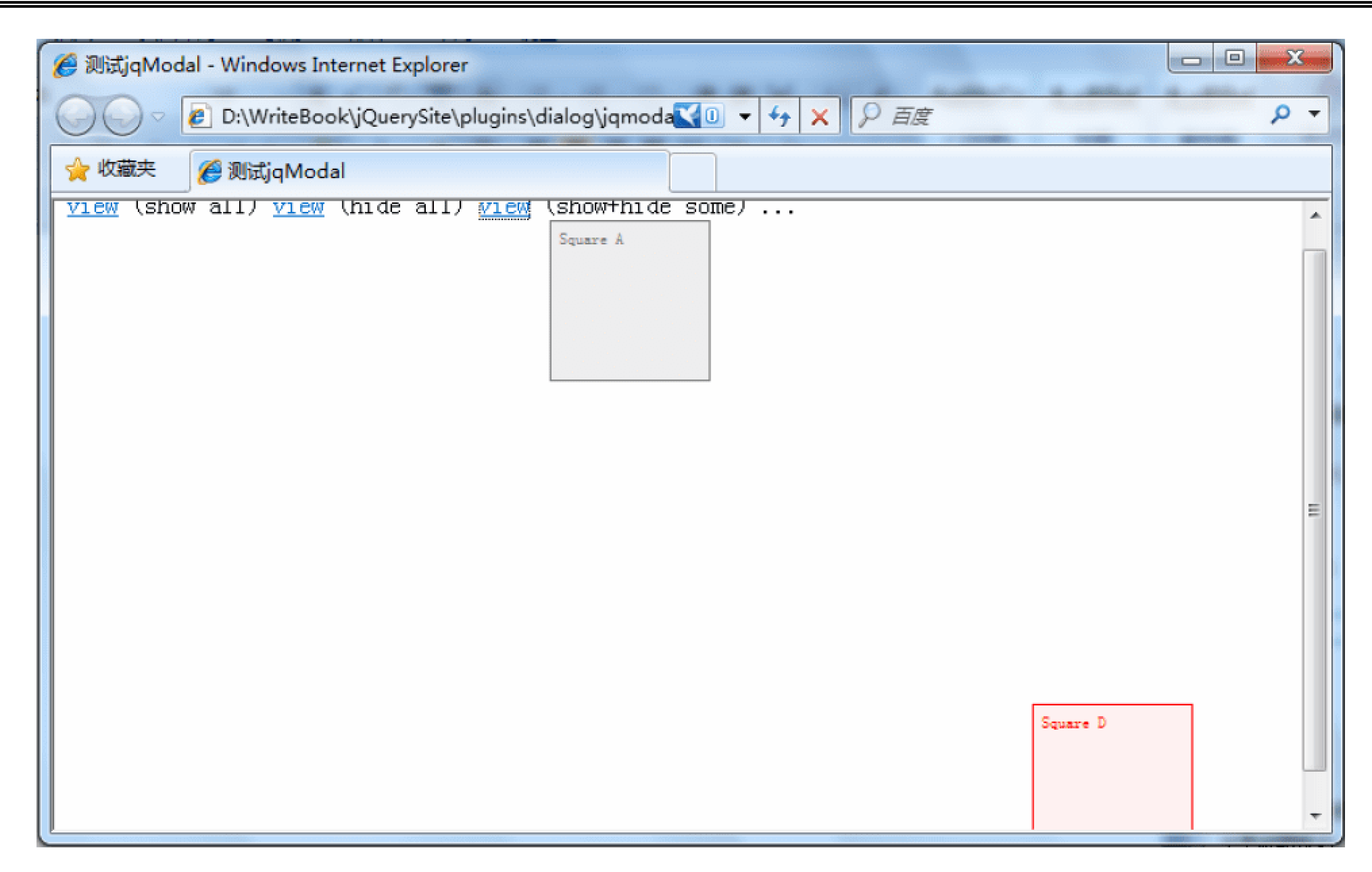


图 9.12　显示部分弹出层

9.6.2　jQuery.UI.Dialog 对话框插件

jQuery.UI.Dialog 对话框插件是 jQuery 官方网站提供的一整套 UI 插件中的一部分。它的主要功能是利用 jQuery 生成动态浮动对话框。这个插件需要以下 jQuery 文件的支持：jquery.ui.core.js、jquery.ui.widget.js、jquery.ui.position.js、jquery.ui.mouse.js（可选，当需要通过鼠标拖动对话框或者改变对话框的大小时使用）、jquery.ui.draggable.js（可选）、jquery.ui.resizable.js。

这个插件有一些相关属性，可以设置对话框的样式和行为，如表 9.3 所示。

表 9.3　Dialog属性说明

属　　性	类　　型	默认值	说　　明
disable	布尔	false	对话框的启动选项，可以在对话框初始化时加载
autoOpen	布尔	true	当对话框被调用的时候自动打开。如果打开失败，则对话框总是处于隐藏状态。dialog("open")表示调用对话框
buttons	对象	{}	指定显示按钮选择。这个属性的键为按钮的文本，值为按钮的单击事件回调函数
buttons	数组	[]	指定显示按钮的选择。数组中的每个元素都必须针对按钮定义
closeOnEscape	布尔	true	设定对话框是否在得到焦点时可以通过 Esc 键关闭
closeText	字符串	close	指定关闭按钮的文本内容
dialogClass	字符串	''	指定初始化对话框的对话框类
draggable	布尔	true	设定对话框是否可以拖动
height	数字	'auto'	对话框的高度，单位为像素
hide	字符串，对象	null	当对话框关闭后的效果
maxHeight	数字	false	对话框可调整的最大高度值，单位为像素
maxWidth	数字	false	对话框可调整的最大宽度值，单位为像素

续表

属　　性	类　　型	默认值	说　　明
minHeight	数字	150	对话框可调整的最小高度值，单位为像素
minWidth	数字	150	对话框可调整的最小宽度值，单位为像素
modal	布尔	false	模式对话框选项
position	字符串，数组	'center'	指定对话框显示的位置：1，单一字符串表示位置'center', 'left', 'right', 'top', 'bottom'；2，两个数字组合数组，单位为像素；3，两个字符串组合的数组
resizable	布尔	true	对话框是否可调整大小选项
show	字符串，对象	null	当对话框打开时的效果
stack	布尔	true	指定当对话框获得焦点时是否可以停靠在多个对话框的最上层
title	字符串	''	对话框的标题属性
width	数字	300	对话框的宽度，单位为像素
zIndex	整型	1000	对话框的初始层位置

与这个对话框相关的还有事件选项，如表 9.4 所示。

表 9.4　Dialog事件说明

事　　件	类　　型	说　　明
create	dialogcreate	对话框创建时触发
beforeClose	dialogbeforeclose	对话框尝试关闭时触发，如果关闭失败，将阻止对话框关闭
open	dialogopen	对话框被打开后触发
focus	dialogfocus	对话框得到焦点时触发
dragStart	dialogdragstart	对话框开始被拖动时触发
drag	dialogdrag	对话框被拖动时触发
dragStop	dialogdragstop	对话框停止拖动时触发
resizeStart	dialogresizestart	对话框开始改变大小时触发
resize	dialogresize	对话框改变大小时触发
resizeStop	dialogresizestop	对话框停止改变大小时触发
close	dialogclose	对话框关闭后触发

有一些行为可以控制该对话框的相关操作，如表 9.5 所示。

表 9.5　Dialog行为说明

行　　为	表　　示	说　　明
destroy	.dialog("destroy")	移除所有对话框的功能
disable	.dialog("disable")	屏蔽对话框
enable	.dialog("disable")	使能对话框
option	.dialog("option" , optionName , [value])	获取或设置对话框参数选项
option	.dialog("option" , options)	设置多个对话框选项
widget	.dialog("widget")	返回对话框元素
close	.dialog("close")	关闭对话框
isOpen	.dialog("isOpen")	如果对话框已经关闭则返回真
moveToTop	.dialog("moveToTop")	移动对话框到页面最上层
open	.dialog("open")	打开对话框

下面通过示例来介绍这个插件的使用方法。

1．默认对话框

这个示例使用 Dialog 的默认参数形式，只调用了 Dialog 插件的初始化函数。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1    <script type="text/JavaScript">
2        $(function() {
3            $( "#dialog" ).dialog();
4        });
5    </script>
```

第 3 行中的.dialog()就是 Dialog 插件的初始化函数。它使用了 Dialog 的默认样式，并在页面加载后自动显示对话框，效果如图 9.13 所示。

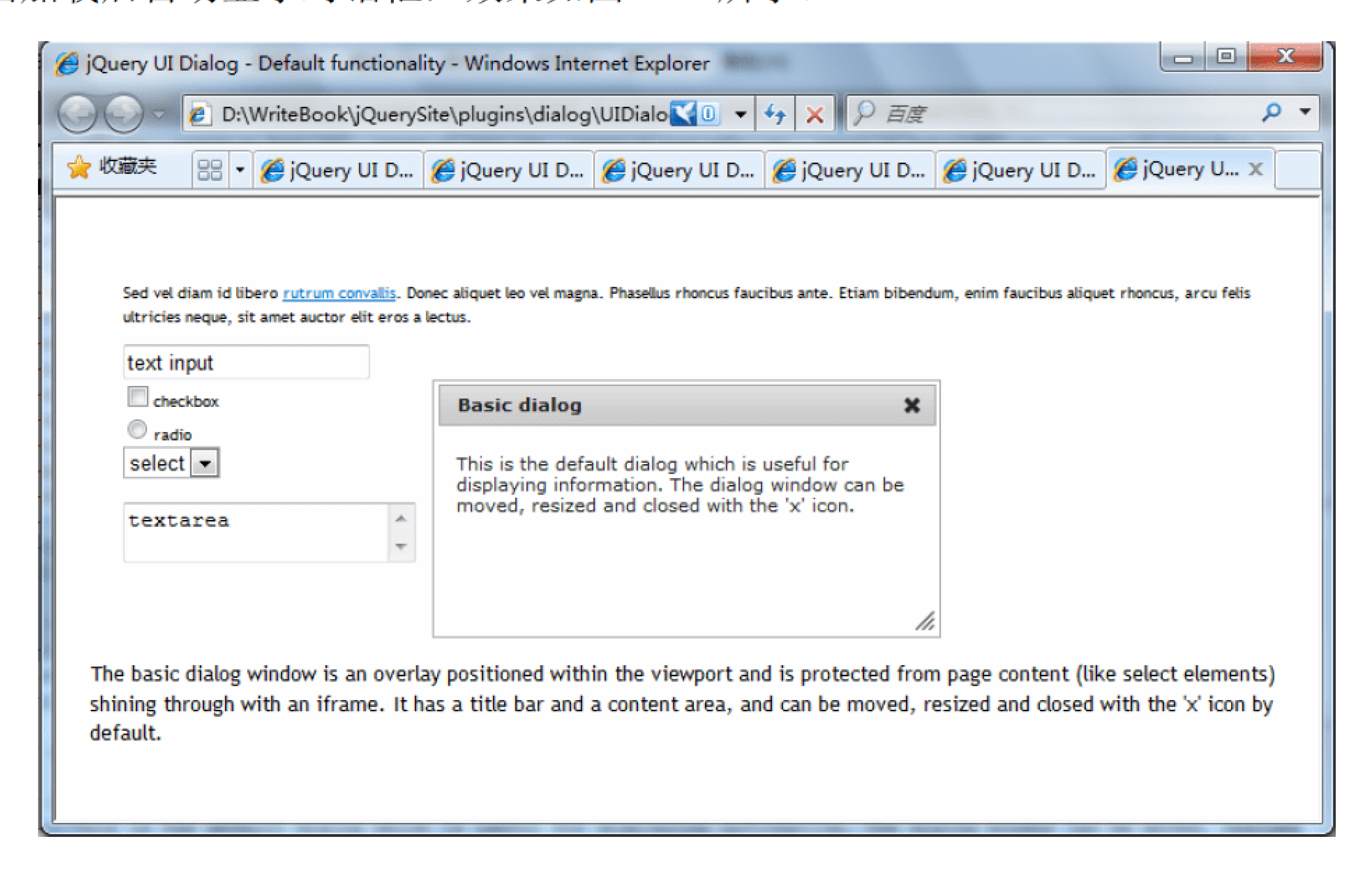


图 9.13　Dialog 插件默认样式

2．动画效果对话框

这个示例使用了关于 Dialog 打开与关闭的属性设置和相关方法调用，实现 Dialog 动画效果打开和关闭。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1    <script>
2    //increase the default animation speed to exaggerate the effect
3        $.fx.speeds. default = 1000;
4        $(function() {
5            $( "#dialog" ).dialog({
6                autoOpen: false,    //不自动打开对话框
7                show: "blind",      //显示样式设定
8                hide: "explode"     //隐藏样式设定
```

```
9            });
10           $( "#opener" ).click(function() {
11               $( "#dialog" ).dialog( "open" );
12               return false;
13           });
14       });
15   </script>
```

上述代码中第 6 行设定对话框不自动打开，第 7 行指定对话框打开时的动画效果为 blind，这个动画效果在 jQuery.effects.blind.js 中定义。第 8 行指定对话框关闭时的动画效果为 explode，这个动画效果在 jQuery.effects. explode.js 中定义。第 10～13 行设定了打开按钮的单击事件，第 11 行使用了 Dialog 的打开行为，具体效果如图 9.14 和图 9.15 所示。当单击 Open Dialog 按钮时，对话框动画渐入打开，当关闭对话框时，对话框分解成 6 个小部分逐渐消失。

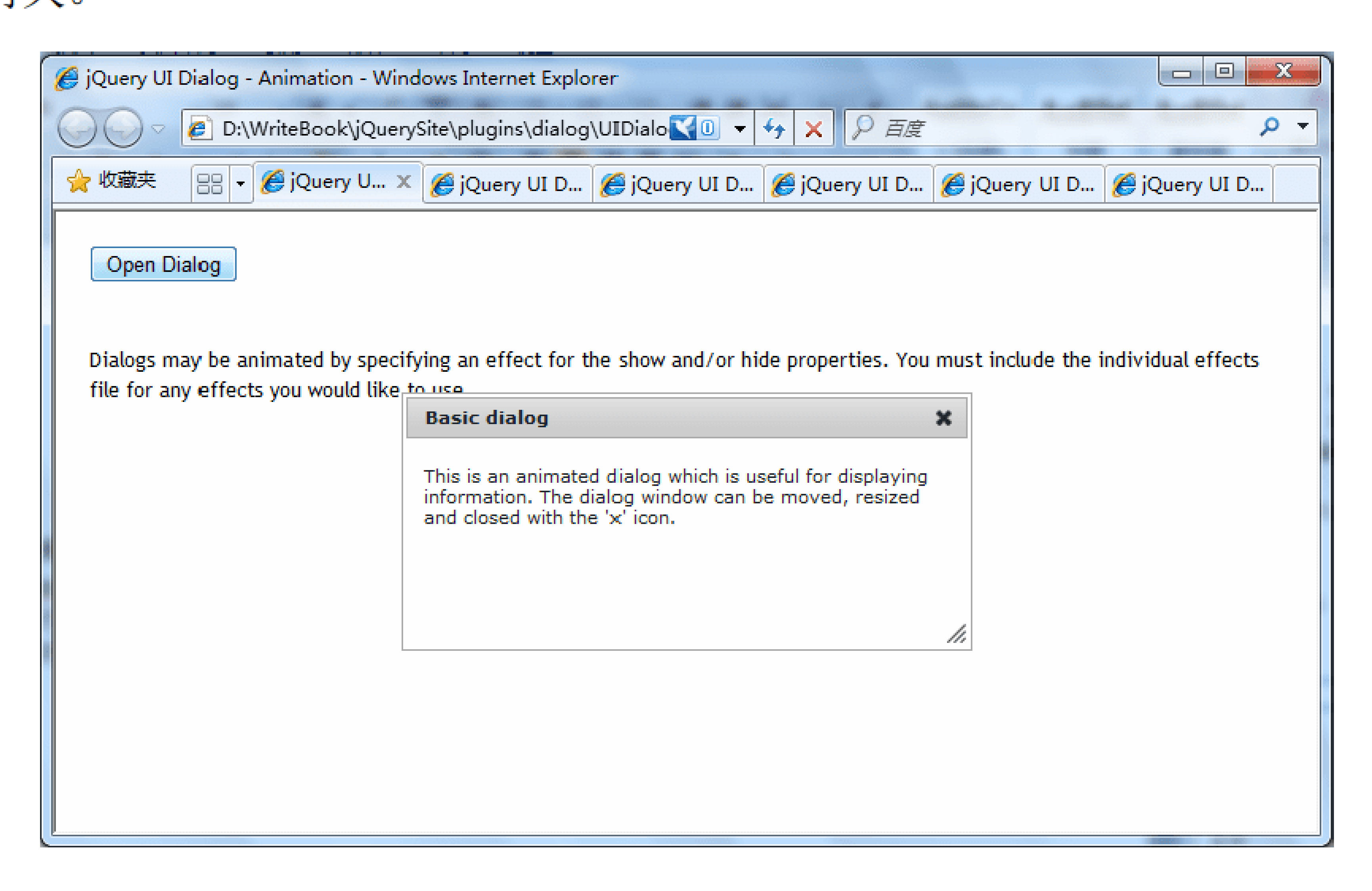


图 9.14　对话框打开

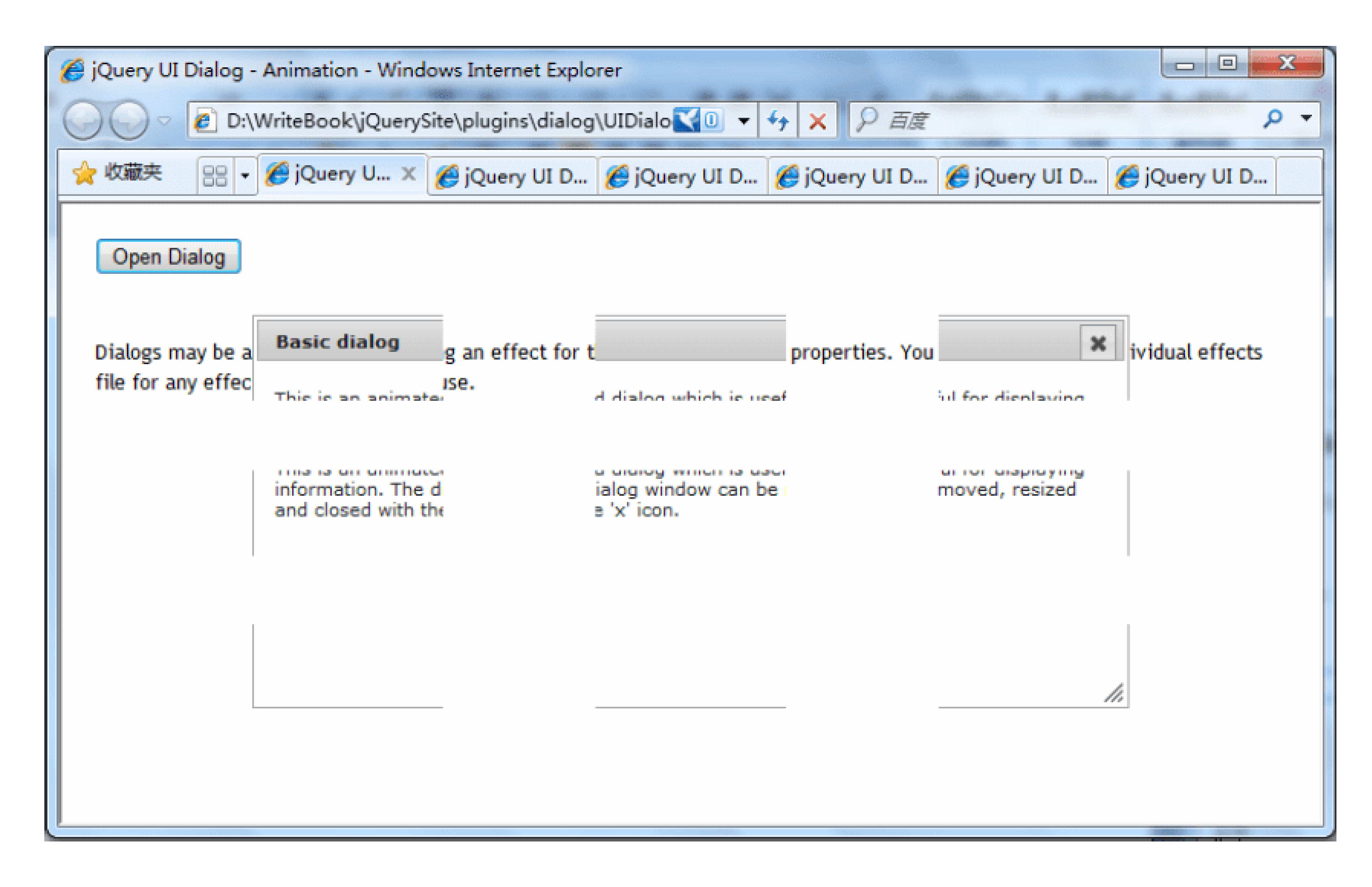


图 9.15　对话框关闭动态过程

3．模式对话框

前面两个示例所看到的对话框都是非模式对话框，也就是当对话框出现时，对话框后面页面上的内容还是可以操作的。下面介绍如何通过 Dialog 创建模式对话框。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1   <script>
2       $(function() {
3           //移除所有对话框功能
4           $( "#dialog:ui-dialog" ).dialog( "destroy" );
5                   $( "#dialog-modal" ).dialog({
6               height: 140,    //设定对话框的高
7               modal: true     //创建模式对话框
8           });
9       });
10  </script>
```

上述代码第 7 行使用了 Dialog 插件的模式对话框属性“modal”，并将其设为真。当页面加载完成后，自动打开一个模式对话框，效果如图 9.16 所示。

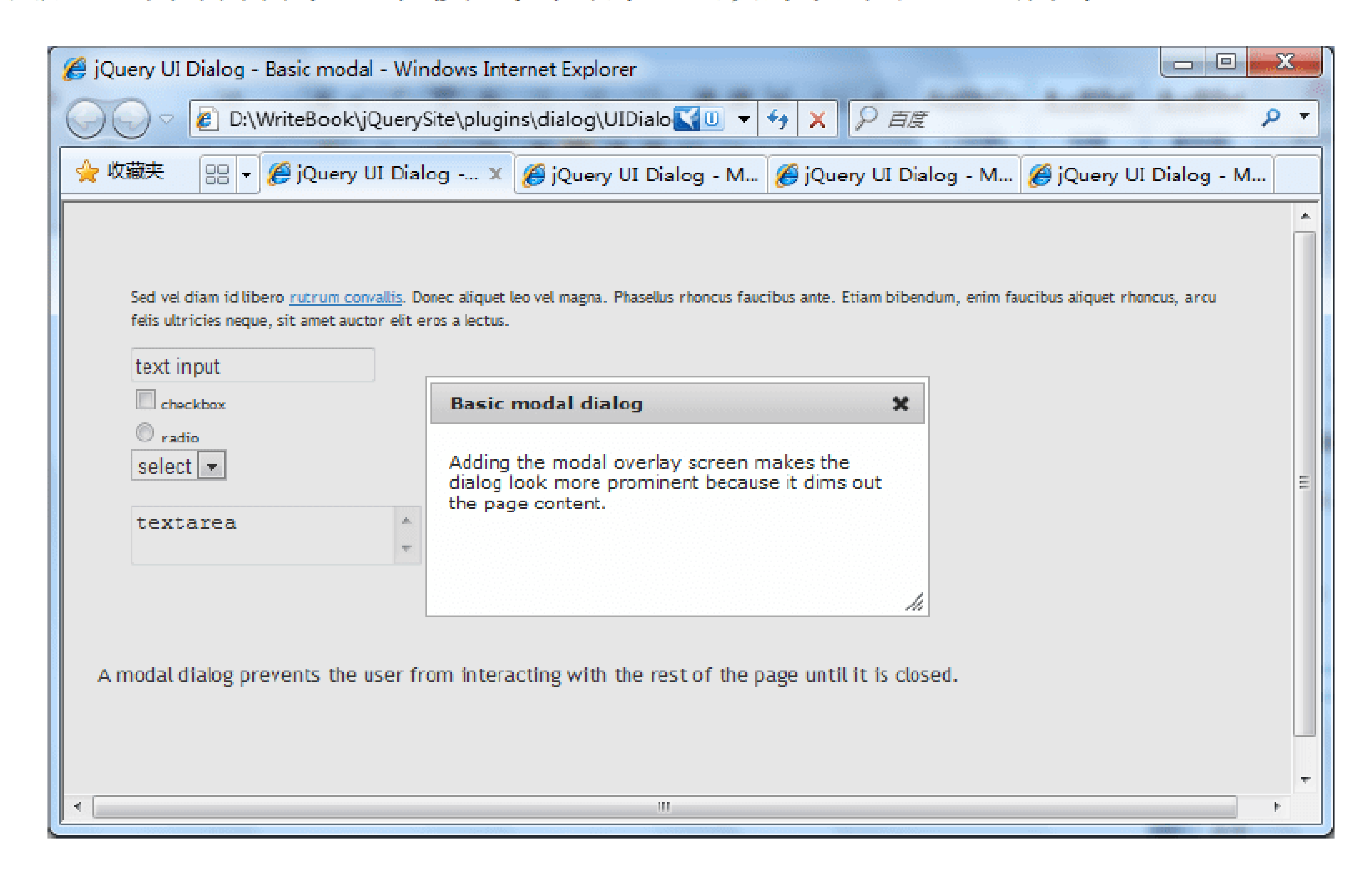


图 9.16　模式对话框

4．模式确认对话框

这个对话框是将原有对话框功能取消掉，利用 Dialog 属性重新创建对话框功能，以实现确认对话框。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       //a workaround for a flaw in the demo system (http://dev.jqueryui.
        com/ticket/4375), ignore!
4       $( "#dialog:ui-dialog" ).dialog( "destroy" );
5       $( "#dialog-confirm" ).dialog({
```

```
6            resizable: false,   //不可调整对话框大小
7            height:140,         //对话框高度
8            modal: true,        //模式对话框
9            buttons: {
10               "Delete all items": function() {
11                   $( this ).dialog( "close" );   //设定对话框功能按钮
12               },
13               Cancel: function() {
14                   $( this ).dialog( "close" );   //对话框“取消”按钮
15               }
16           }
17       });
18   });
19   </script>
```

上述代码第 4 行销毁原有对话框的所有默认属性。第 6 行设定对话框禁止调整大小。第 7 行设定对话框高度为 140 像素。第 8 行设定模式对话框。第 9～16 行添加按钮功能。第 10 行指定文本为“Delete all items”的按钮的关闭功能。第 11 行指定了对话框的关闭行为。第 13 行同样指定了“取消”按钮的对话框关闭行为。效果如图 9.17 所示。

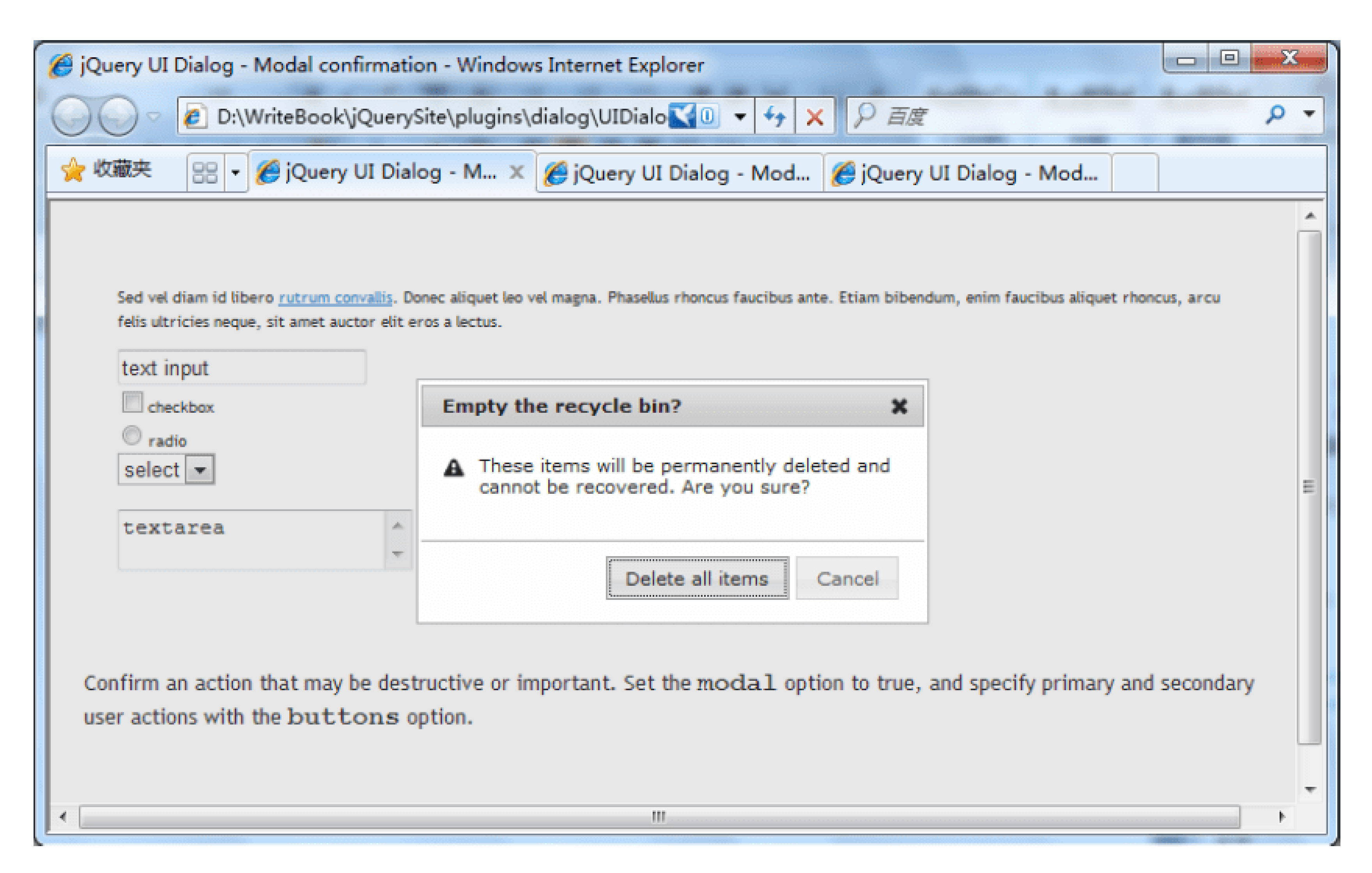


图 9.17　确认模式对话框

5．模式表单对话框

这个对话框是将原有对话框功能取消掉，利用 Dialog 属性重新创建对话框功能，以实现用户输入表单对话框。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1   $(function() {
2       //移除所有对话框功能
3       $( "#dialog:ui-dialog" ).dialog( "destroy" );
4
```

```
5       var name = $( "#name" ),
6           email = $( "#email" ),
7           password = $( "#password" ),
8           allFields = $( [] ).add( name ).add( email ).add( password ),
9           tips = $( ".validateTips" );
10      function updateTips( t ) {
11          tips
12              .text( t )
13              .addClass( "ui-state-highlight" );
14          setTimeout(function() {
15              tips.removeClass( "ui-state-highlight", 1500 );
16          }, 500 );
17      }
18      function checkLength( o, n, min, max ) {
19          if ( o.val().length > max || o.val().length < min ) {
20              o.addClass( "ui-state-error" );
21              updateTips( "Length of " + n + " must be between " +
22                  min + " and " + max + "." );
23              return false;
24          } else {
25              return true;
26          }
27      }
28      function checkRegexp( o, regexp, n ) {
29          if ( !( regexp.test( o.val() ) ) ) {
30              o.addClass( "ui-state-error" );
31              updateTips( n );
32              return false;
33          } else {
34              return true;
35          }
36      }
37      $( "#dialog-form" ).dialog({
38          autoOpen: false,
39          height: 300,
40          width: 350,
41          modal: true,
42          buttons: {
43              "Create an account": function() { //实现表单信息添加功能按钮
44                  var bValid = true;
45                  allFields.removeClass( "ui-state-error" );
46                  bValid = bValid && checkLength( name, "username", 3, 16 );
47                  bValid = bValid && checkLength( email, "email", 6, 80 );
48                  bValid = bValid && checkLength( password, "password", 5, 16 );
49                  bValid = bValid && checkRegexp( name, /^[a-z]([0-9a-z ])
                    +$/i, "Username may consist of a-z, 0-9, underscores,
                    begin with a letter." );
                    //From jquery.validate.js (by joern), contributed by Scott
```

```
                    Gonzalez: http://projects.scottsplayground.com/email_
                    address_validation/
                    bValid = bValid && checkRegexp( email, /^((([a-z]
                    |\d|[!#\$%&'\*\+\-\/=\?\^_`{\|}~]|[\u00A0-\uD7FF
                    \uF900-\uFDCF\uFDF0-\uFFEF])+(\.([a-z]|\d|[!#\$%&'\
                    *\+\-\/=\?\^_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-
                    \uFFEF])+)*)|((\x22)((((\x20|\x09)*(\x0d\x0a))?(\x20|
                    (([\x01-\x08\x0b\x0c\x0e-\x1f\x7f]|\x21|\x09)+)?[\x23-
                    \x5b]|[\x5d-\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\
                    uFFEF])|(\\([\x01-\x09\x0b\x0c\x0d-\x7f]|[\u00A0-\uD7FF\
                    uF900-\uFDCF\uFDF0-\uFFEF]))))*(((\x20|\x09)*(\x0d\x0a))?
                    (\x20|\x09)+)?(\x22)))@((([a-z]|\d|[\u00A0-\uD7FF\
                    uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|\d|[\u00A0-\uD7FF\
                    uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|[\u00A0-
                    \uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|\d|[\u00A0-
                    \uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.)+(([a-z]|[\u00A0-
                    \uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-z]|[\u00A0-
                    \uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|_|~|
                    [\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|
                    [\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.?$/i,
                    "eg. ui@jquery.com" );
50                  bValid = bValid && checkRegexp( password, /^([0-9a-zA-
                    Z])+$/, "Password field only allow : a-z 0-9" );
51                  if ( bValid ) {
52                      $( "#users tbody" ).append( "<tr>" +
53                          "<td>" + name.val() + "</td>" +
54                          "<td>" + email.val() + "</td>" +
55                          "<td>" + password.val() + "</td>" +
56                      "</tr>" );
57                      $( this ).dialog( "close" );
58                  }
59              },
60              Cancel: function() {
61                  $( this ).dialog( "close" );
62              }
63          },
64          close: function() {
65              allFields.val( "" ).removeClass( "ui-state-error" );
66          }
67      });
68      $( "#create-user" )
69          .button()
70          .click(function() {
71              $( "#dialog-form" ).dialog( "open" );
72          });
73  });
74  </script>
```

上述代码第 3 行取消对话框原有功能。第 5～9 行获取对话框表单元素对象及验证提示信息显示对象。第 10～17 行根据验证的错误更新验证信息显示内容。第 18～27 行检查每个表单元素的用户输入长度。第 28～36 行进行正则表达式验证。第 37 行初始化插件。第 38 行设定对话框不自动打开。第 39、40 行设定对话框宽 350 像素、高 300 像素。第 41 行设定对话框为模式对话框。第 42～66 行设定对话框相关按钮的功能。

其中第 43～59 行为文本为“Create an account”的按钮设定表单元素内容验证功能。如果验证通过，则在页面上的表格中添加一条用户信息，否则在验证消息部分显示错误消息。第 60～62 行设定“取消”按钮的关闭对话框功能。第 64～66 行设定“关闭”按钮的附加功能，将表单元素内容设置为空，并取消验证错误样式。

第 68～72 行添加调用打开对话框行为的按钮功能，效果如图 9.18～图 9.20 所示。

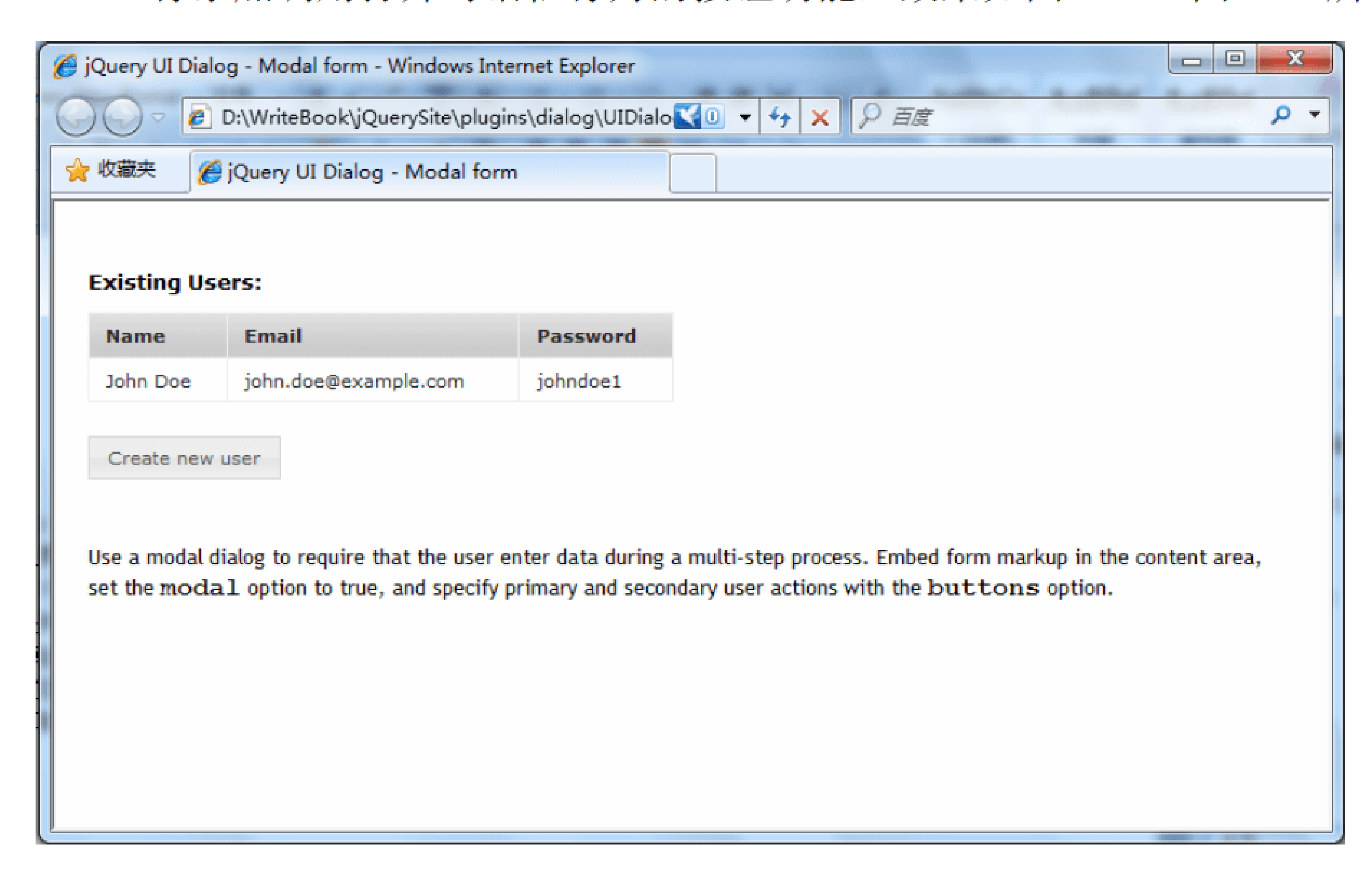


图 9.18　页面加载后效果

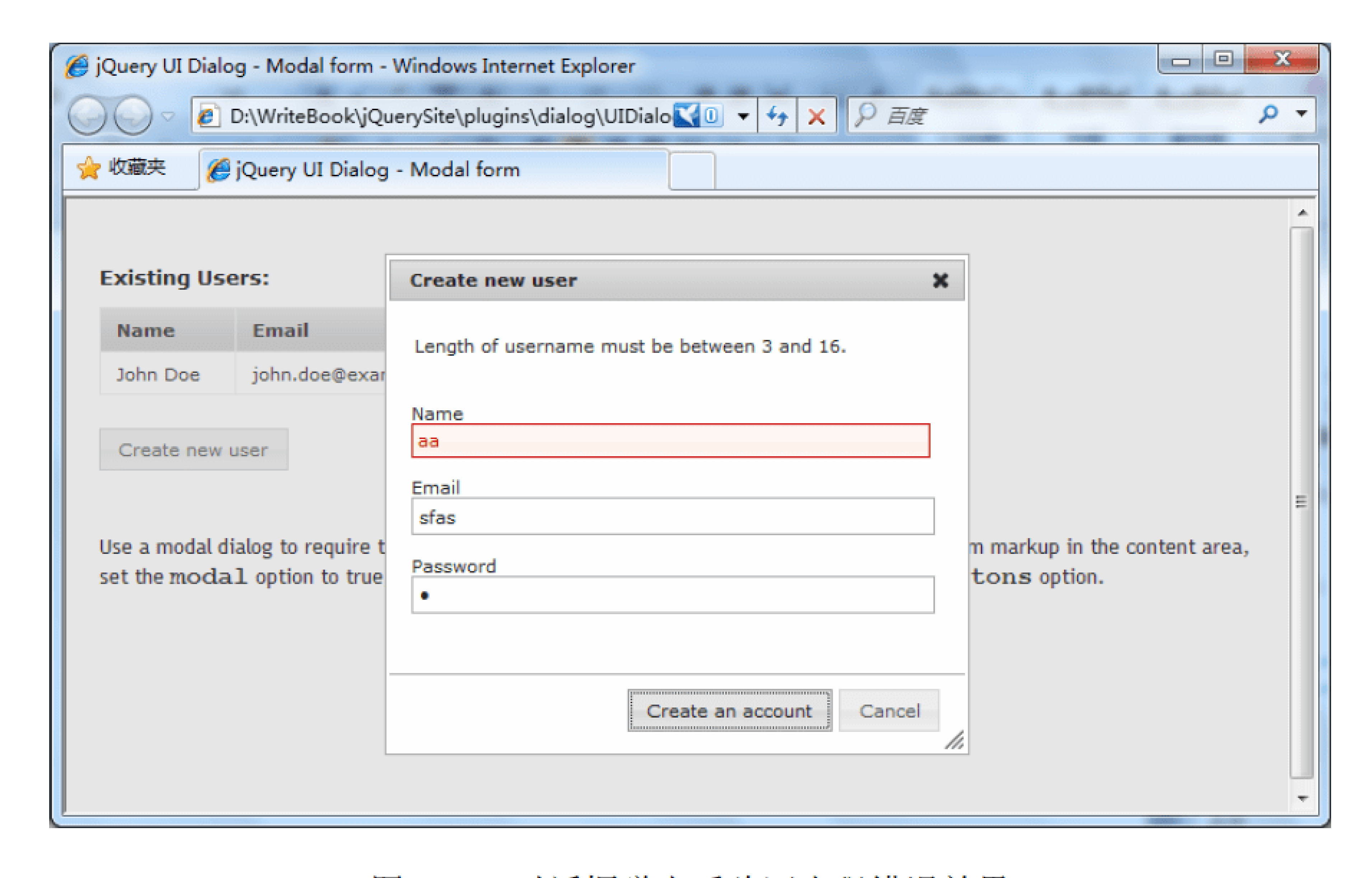


图 9.19　对话框弹出后验证出现错误效果

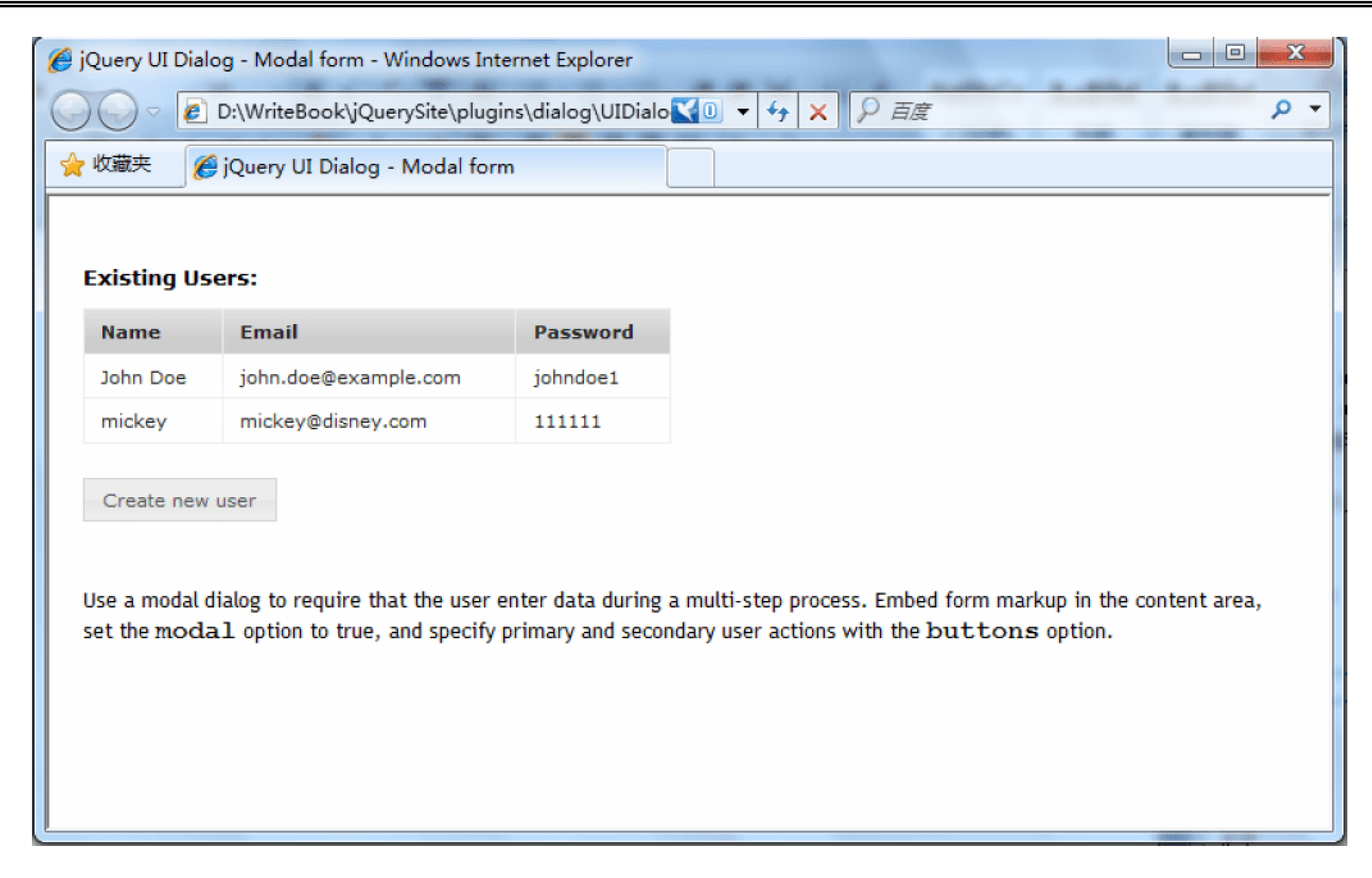


图 9.20　在对话框中成功填入信息后的页面效果

6．模式消息对话框

这个对话框取消原有对话框功能，利用 Dialog 属性重新创建对话框功能，并在对话框中显示一定的消息内容。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       //a workaround for a flaw in the demo system (http://dev.
        jqueryui.com/ticket/4375), ignore!
4       $( "#dialog:ui-dialog" ).dialog( "destroy" );
5       $( "#dialog-message" ).dialog({
6           modal: true,
7           buttons: {
8               Ok: function() {
9                   $( this ).dialog( "close" );
10              }
11          }
12      });
13  });
14  </script>
```

上述代码第 4 行的功能与前面的示例相同。第 6 行设定模式对话框形式。第 8、9 行为 OK 按钮添加对话框的关闭功能。效果如图 9.21 所示。

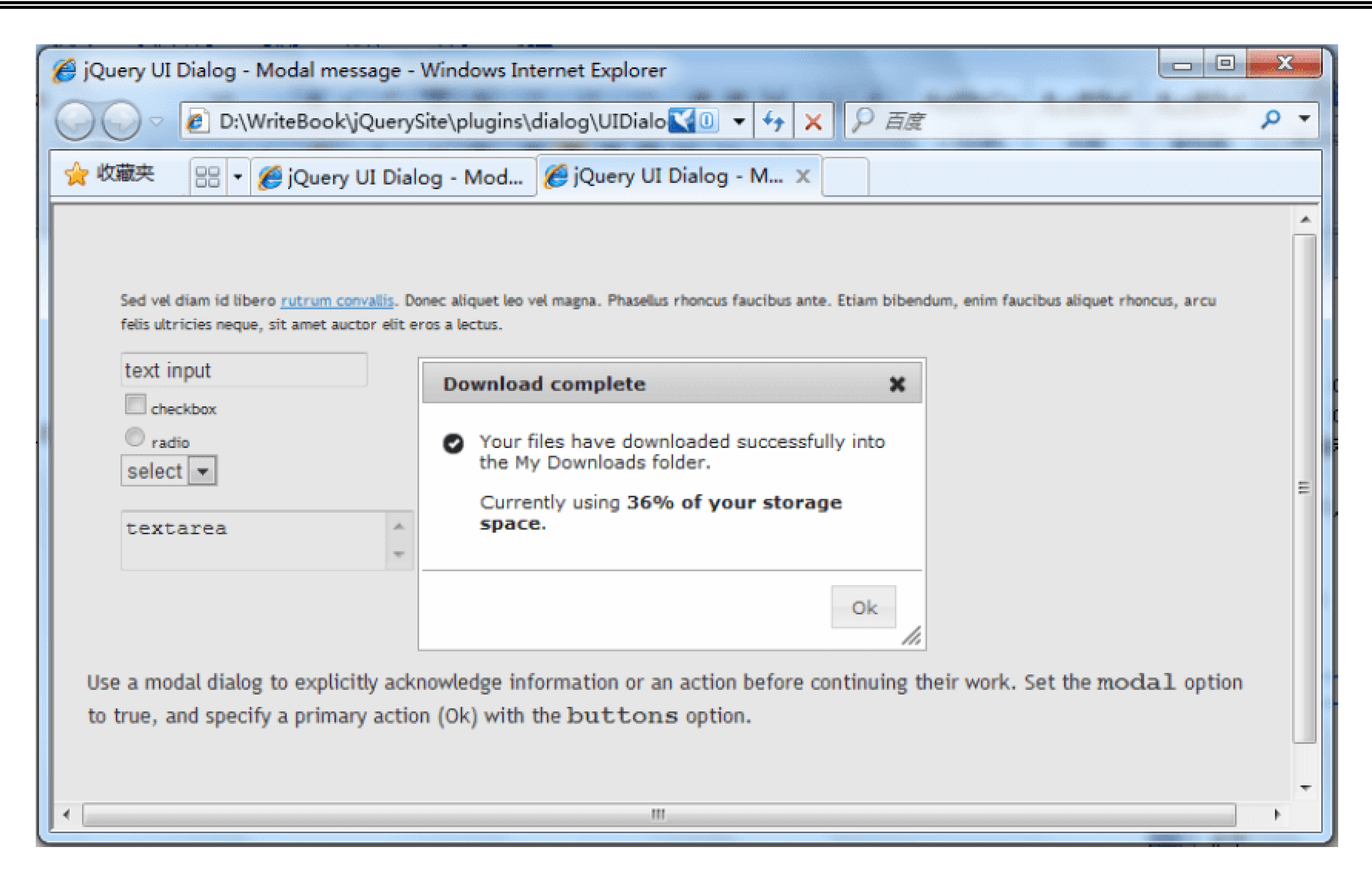


图 9.21　模式消息对话框

9.7　小　　结

本章对 jQuery 创建网页对话框进行了讲解。主要内容包括基本对话框实现原理，模式对话框及常用对话框的创建及操作，并介绍了两个对话框插件。重点部分是对话框的实现原理及 Dialog 插件的使用，同时这一部分也是本章的难点。下一章将讲解 jQuery 对滑动条的实现。

9.8　习　　题

【习题 1】利用本章所学内容自定义一个带表单录入功能的模式对话框。
【习题 2】练习 jQuery.UI.Dialog 对话框插件的使用方法。
【习题 3】练习 jqModal 对话框插件的使用方法。

第 10 章　设计滑动条

在很多网页中，都会看到页面中的部分区域可以使用滑动条操作显示内容，但是这里的滑动条与浏览器的滑动条不同。这种滑动条其实也是页面元素的一部分，可以通过程序来控制其出现和如何操作。本章主要讲解如何使用 jQuery 创建及操作滑动条。

10.1　自定义滑动条

本节主要讲解如何使用 jQuery 创建区域滑动条。它的实现原理是：在页面的内容显示区域中，创建长短适当的滑动条层，并在滑动条层上添加鼠标移动和鼠标键释放事件，以产生内容和移动条上下移动的效果。其中，使用到了 jQuery 函数 ready()、height()、show()、css()、find()、mousemove()、mousedown()。

其功能实现如下。

（1）判断内容所在层的高度是否大于显示区域的预定高度。如果大于，则加入滑动条，否则不添加滑动条。

（2）加载滑动条效果。

（3）设定滑动条区间的鼠标按下事件，在事件中提取鼠标位置。

（4）设定滑动块鼠标移动事件，判断鼠标移动位置，并按照鼠标移动的合理位置移动显示内容。

首先，利用 HTML 创建静态页面。相关的 CSS 代码请参考光盘内容。

```
1 <div id="subwindow">
2   <div id="scrollarea">
3     <div id="scrollblock">
4       <img id="scrollimg" src=""/><br />
5       </div>
6   </div>
7   <div id="content">
8       <img src="img/img8.gif" />
9   </div>
10 </div>
```

然后，引入 jQuery 库文件：

```
<script type="text/JavaScript" src="jslib/jquery.js"></script>
```

最后，加入 JavaScript 功能代码：

```
1 <script type="text/JavaScript">
2   $(function(){
3       if($("#content").height()>$("#subwindow").height())
4       {
5           $("#scrollarea").show("slow"); //显示滚动区域
            //调整滚动块的高度，单位为像素，值为整数
6   $("#scrollblock").find("img").height(parseInt($("#subwindow").
    height()*$("#subwindow").height()/ $("#content").height()));
7           $("#scrollblock").show("slow");//显示滚动块
            //定义鼠标位置变量及内容坐标变量
8           var mouseX=0;
9           var mouseY=0;
10          var contentX=0;
11          var contentY=0;
12          $("#scrollarea").mousedown(function(e){
                                    //鼠标在滚动区间内的左键按下事件
13              var devent=e||event;        //获取事件参数
14              mouseX=devent.clientX;      //获取鼠标坐标
15              mouseY=devent.clientY;
16              contentX=parseInt($("#scrollblock").css("left"));
                                    //获取内容位置
17              contentY=parseInt($("#scrollblock").css("top"));

18          });
19          $("#scrollblock").mousemove(function(e){    //鼠标移动滚动块事件
20              var devent=e||event;                    //获取事件参数
21              var distance=devent.clientY-mouseY+contentY;
                                    //计算鼠标纵向移动影响内容移动位移
22              if(distance<0)
23                  distance=0;
24              else if(distance>=$("#scrollarea").height()-$
                ("#scrollblock").height())
25                 distance=$("#scrollarea").height()-$("#scrollblock").
                    height();
26              $("#scrollblock").css("top",distance+"px");
                                    //修改滑动块纵向位移
27  $("#content").css("top",-(distance*(parseInt($("#content").height()
    -$("#subwindow").height()))/(parseInt($("#scrollarea").height())-
    parseInt($("#subwindow").height()*$("#subwindow").height()/ $
    ("#content").height()))))+"px");
28          });             //更改内容移动位移
29
30      }
31  });
32 </script>
```

上述代码第 5～7 行对滑动条的显示进行设置。第 12～18 行设定滑动条区域的鼠标按下事件。在这个事件中获取了鼠标位置和滑动块的位置。第 19～28 行设定了滑动块的鼠标移动事件，判断鼠标的偏移位移是否在合理区间内。如果滑动块可以移动，则移动滑动块，并移动对应位移的内容。效果如图 10.1 所示。

图 10.1　自定义滑动块

10.2　滑动条插件

本节介绍几种滑动条插件。这些插件为页面创建出了不同样式、不同功能的滑动条效果。

10.2.1　jQuery.UI.Slider 插件

jQuery.UI.Slider 插件是 jQuery 官方网站提供的一整套 UI 插件中的一部分。它的主要功能是利用 jQuery 生成特定功能的滑动条。这个插件需要以下 jQuery 文件的支持：jquery.ui.core.js、jquery.ui.widget.js、jquery.ui.mouse.js。

这个插件有一些相关属性，可以设置滑动条的样式和行为，如表 10.1 所示。

表 10.1　滑动条的属性

属　性	类　型	默 认 值	说　明
disable	布尔	false	屏蔽或者启用滑动条功能
animate	布尔，字符串，数字	false	当用户单击滑动块外部时，它的处理效果是否平滑；或者使用不同字符串表示动画速度；也可以用数字表示动画具体执行毫秒数
max	数字	100	滑动条所能表示的最大值
min	数字	0	滑动条所能表示的最小值
orientation	字符串	'horizontal'	滑动条的滑动方向，是从左向右，还是从下向上
range	布尔，字符串	false	范围设定，如果设置为真，则检测是否有两个滑动块，一块从小向大滑动，另一块从大向小移动
step	数字	1	滑动块移动的步长或者单位间隔
value	数字	0	设置滑动块的值，如果有多个滑动块，则对第一个进行设置
values	数组	null	设置多个滑动块的值

与这个滑动条相关的还有事件选项，如表 10.2 所示。

表 10.2　滑动条的事件

事　件	类　型	说　明
create	slidecreate	滑动条被创建时触发
start	slidestart	当用户开始滑动滑动块时触发
slide	slide	当鼠标移动滑动块时触发
change	slidechange	滑动操作停止时触发，或者通过编程改变当前值时触发
stop	slidestop	当用户停止滑动操作时触发

滑动条有一些行为可以控制滑动条的相关操作，如表 10.3 所示。

表 10.3　滑动条的行为

行　　为	表　　示	说　　明
destroy	.slider("destroy")	完全删除滑动条的功能
disable	.slider("disable")	屏蔽滑动条
enable	.slider("enable")	启用滑动条
option	.slider("option" , optionName , [value])	获取或设置滑动条属性
widget	.slider("widget")	返回滑动条元素
value	.slider("value" , [value])	获取或设置单滑动块的滑动条的值
option	.slider("option" , options)	一次设定多个滑动条属性
values	.slider("values" , index , [value])	获取或设置多个滑动块的滑动条的所有值

下面通过示例来介绍一下这个插件的使用方法。

1．默认样式滑动条

这个示例使用滑动条的默认参数形式，只调用了滑动条插件的初始化函数。这种情况下滑动块可以随意移动，并且移动的单位长度也不固定。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       $( "#slider" ).slider();    //利用滑动条插件基本功能
4   });
5   </script>
```

第 3 行中的 slider()就是滑动条插件的初始化函数。它使用了滑动条的默认样式，并在页面加载后自动显示滑动条，效果如图 10.2 所示。

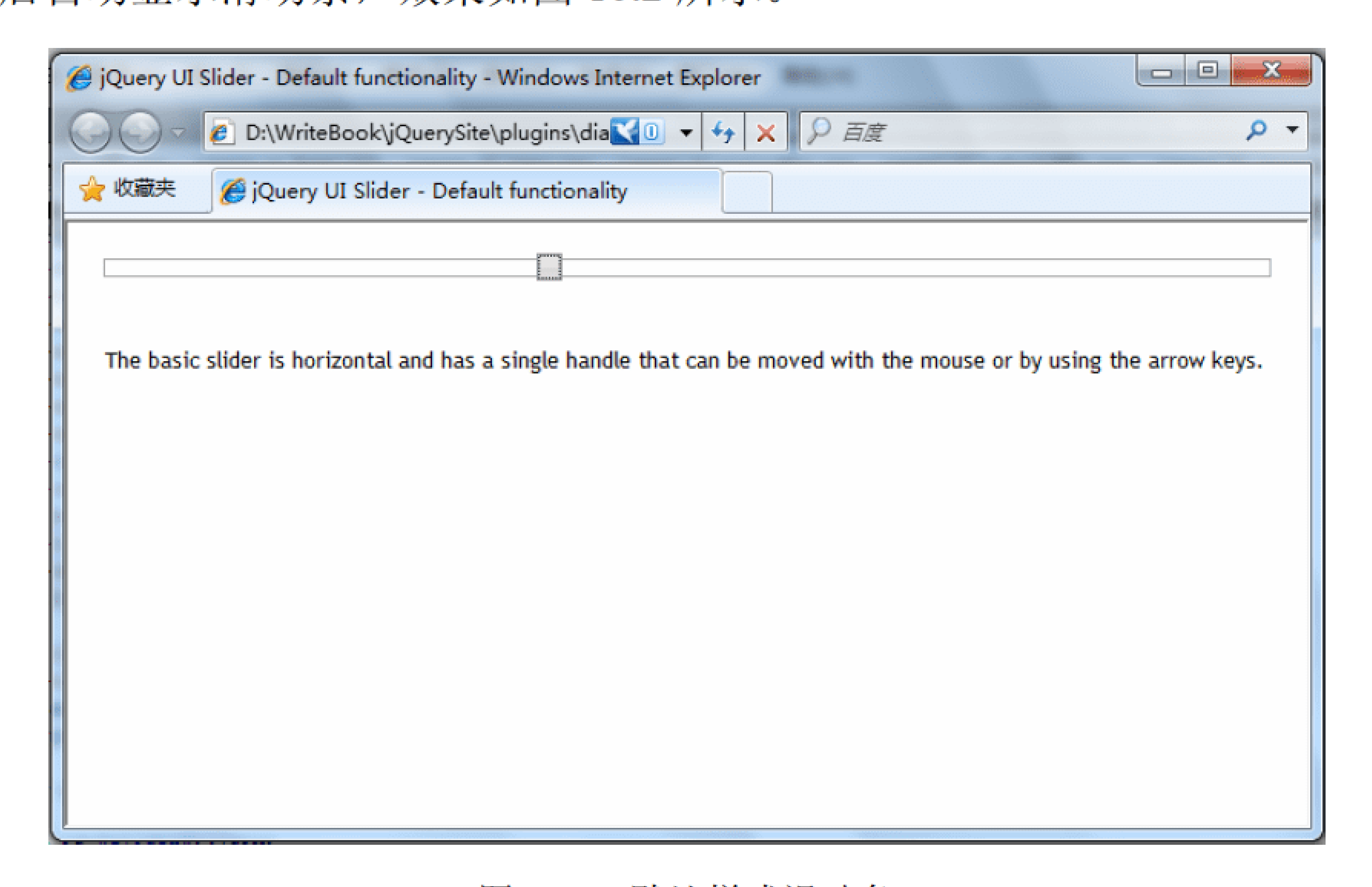


图 10.2　默认样式滑动条

2．控制步进滑动条

这个示例使用前面介绍的滑动条的步进属性（即滑动条移动的单位长度），并使用滑

动条插件的初值及最大最小值属性和滑动事件。与前面的示例不同的是滑动块具有了最大和最小移动范围，并且单位移动长度已知，便于我们通过判断滑动块的位置而得到它所代表的值。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       $( "#slider" ).slider({
4           value:100,                      //滑动条初始位置
5           min: 0,                         //滑动条最小表示位置
6           max: 500,                       //滑动条最大表示位置
7           step: 50,                       //滑动条单位移动位移
8           slide: function( event, ui ) { //滑动条滑动事件处理
9               $( "#amount" ).val( "$" + ui.value );
10          }
11      });
12      $( "#amount" ).val( "$" + $( "#slider" ).slider( "value" ) );
13  });
14  </script>
```

上述代码中第 4 行设定滑动块的初始位置。第 5 行设定滑动条最小值。第 6 行设定最大值。第 7 行设定步进为 50。第 9、10 行设定当滑动块移动时更改显示值。效果如图 10.3 所示。

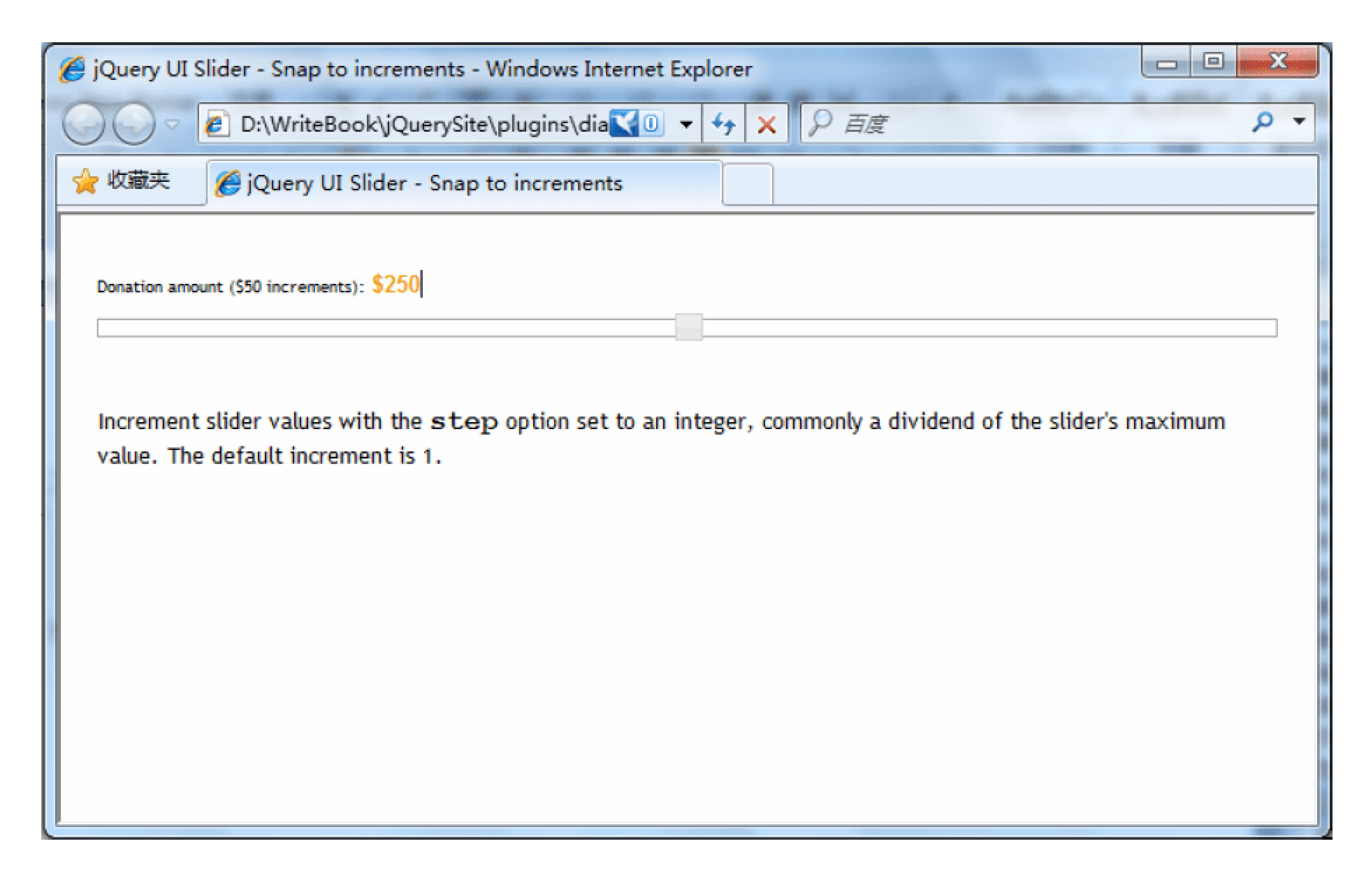


图 10.3　设定步进滑动条

3．设定表示范围的滑动条

这个示例使用滑动条的范围参数，并使用滑动条插件的初值及最大最小值属性和滑动事件。表示范围的滑动框由分别表示范围上限和范围下限的两个滑动块组成，并且每个滑动块都有自己的移动区间，用户可以更形象地通过移动滑动块来获取范围空间。JavaScript 功能代码如下：

```
1    <script>
2    $(function() {
3        $( "#slider-range" ).slider({
4            range: true,                    //设定表示范围功能
5            min: 0,                         //范围最小值可为 0
6            max: 500,                       //范围最大值为 500
7            values: [ 75, 300 ],            //表示范围区间的初始值
8            slide: function( event, ui ) { //滑动条滑动事件
9                $( "#amount" ).val( "$" + ui.values[ 0 ] + " - $" + ui.
                 values[ 1 ] );
10           }
11       });
12       $( "#amount" ).val( "$" + $( "#slider-range" ).slider( "values", 0 ) +
13           " - $" + $( "#slider-range" ).slider( "values", 1 ) );
14   });
15   </script>
```

上述代码中第 4 行设定了滑动条功能为表示范围，即具有两个滑动块，分别表示最大值和最小值。第 7 行设定了范围初始值，即两个滑动块的初始位置。效果如图 10.4 所示。

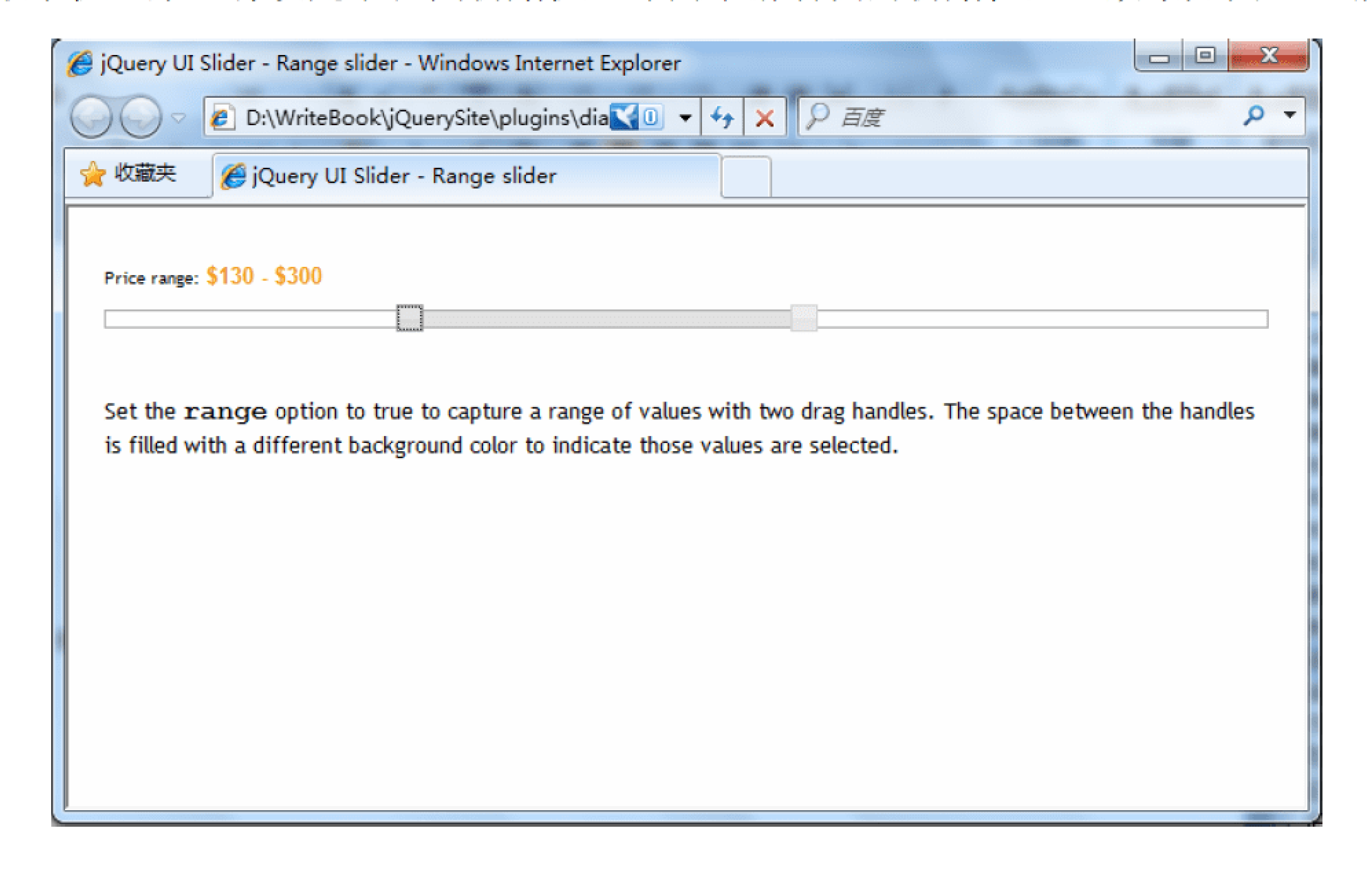


图 10.4　设定范围滑动条

4．固定了最小值范围的滑动条

这个示例使用滑动条的范围参数，并使用滑动条插件的初值及最大最小值属性和滑动事件。固定最小值滑动块的位置即固定了范围下限，通过代表范围上限的滑动块的移动获取区间值。JavaScript 功能代码如下：

```
1    <script>
2    $(function() {
3        $( "#slider-range-min" ).slider({
```

```
4           range: "min",                    //设定最小值范围
5           value: 37,                       //最小范围值为 37
6           min: 1,                          //最小值为 1
7           max: 700,                        //最大值为 700
8           slide: function( event, ui ) {   //滑动条滑动事件
9               $( "#amount" ).val( "$" + ui.value );
10          }
11      });
12      $( "#amount" ).val( "$" + $( "#slider-range-min" ).slider
        ( "value" ) );
13  });
14  </script>
```

上述代码第 4 行设定滑动条功能为表示范围，但固定了范围最小值不动，用户只可修改范围最大值。效果如图 10.5 所示。

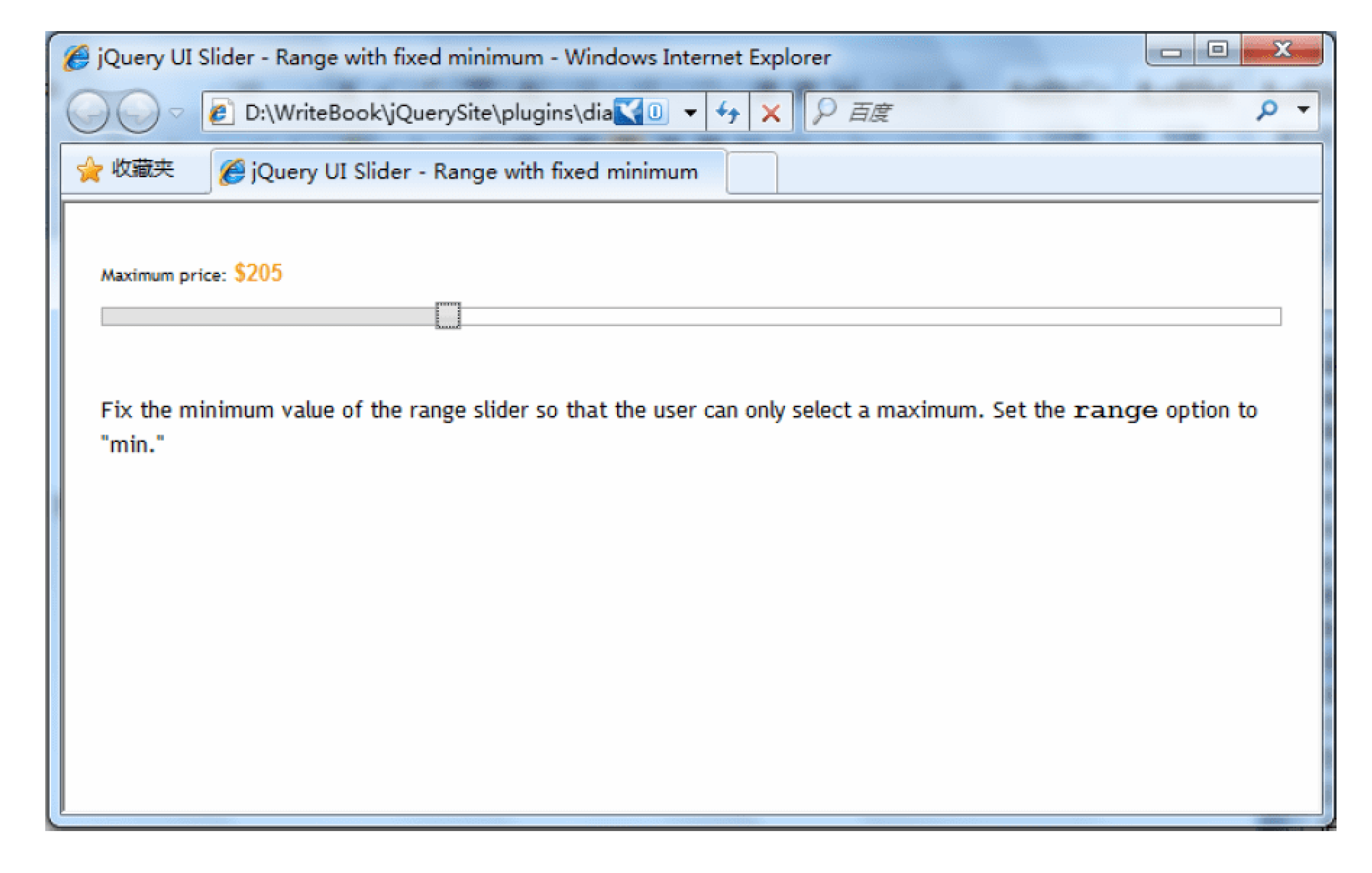


图 10.5　固定了最小值的范围滑动条

5. 固定最大值的滑动条

这个示例使用滑动条的范围参数，并使用滑动条插件的初值及最大最小值属性和滑动事件。与上例相反，此处是下限可调整。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       $( "#slider-range-max" ).slider({
4           range: "max",                    //设定最大范围表示功能
5           min: 1,                          //最小表示值为 1
6           max: 10,                         //最大表示值为 10
7           value: 2,                        //初始值为 2
8           slide: function( event, ui ) {   //滑动条滑动事件
9               $( "#amount" ).val( ui.value );
10          }
11      });
```

```
12      $( "#amount" ).val( $( "#slider-range-max" ).slider( "value" ) );
13  });
14  </script>
```

上述代码第 4 行设定滑动条功能为表示范围，但固定了范围最大值不动，用户只可修改范围最小值。效果如图 10.6 所示。

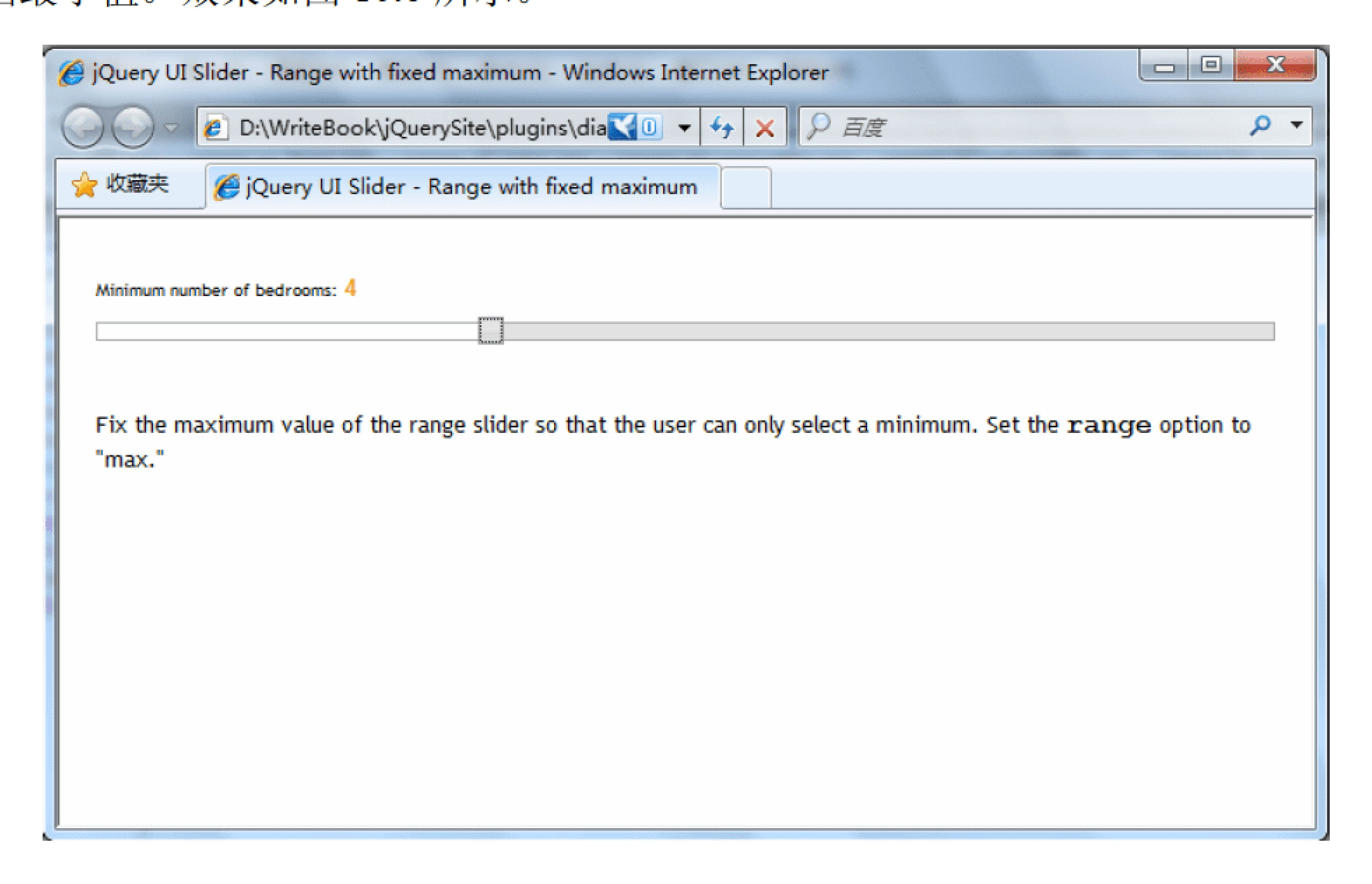


图 10.6　固定了最大值的范围滑动条

6．通过外部元素改变滑动条

这个示例使用滑动条的范围参数，并使用滑动条插件的初值及最大最小值属性和滑动事件。通过用户在其他部分输入范围值，并将此值在滑动块上显示出来，使用户更形象地了解自己选择的范围值。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       var select = $( "#minbeds" );
4       var slider = $( "<div id='slider'></div>" ).insertAfter
        ( select ).slider({
5           min: 1,                             //最小表示值为 1
6           max: 6,                             //最大表示值为 6
7           range: "min",                       //设定最小范围表示功能
8           value: select[ 0 ].selectedIndex + 1,
9           slide: function( event, ui ) {      //滑动条滑动事件
10              select[ 0 ].selectedIndex = ui.value - 1;
11          }
12      });
13      $( "#minbeds" ).change(function() {
14          slider.slider( "value", this.selectedIndex + 1 );
15      });
16  });
17  </script>
```

上述代码第 4 行通过 jQuery 动态创建了滑动条所在层，并调用滑动条初始化函数。第 8 行对表示范围上限的滑动块设定初始值，但是这个值是通过下拉列表框获取的。这个滑动条也是范围滑动条，固定了最小值不变化，并可通过下拉列表框修改最大值。效果如图 10.7 所示。

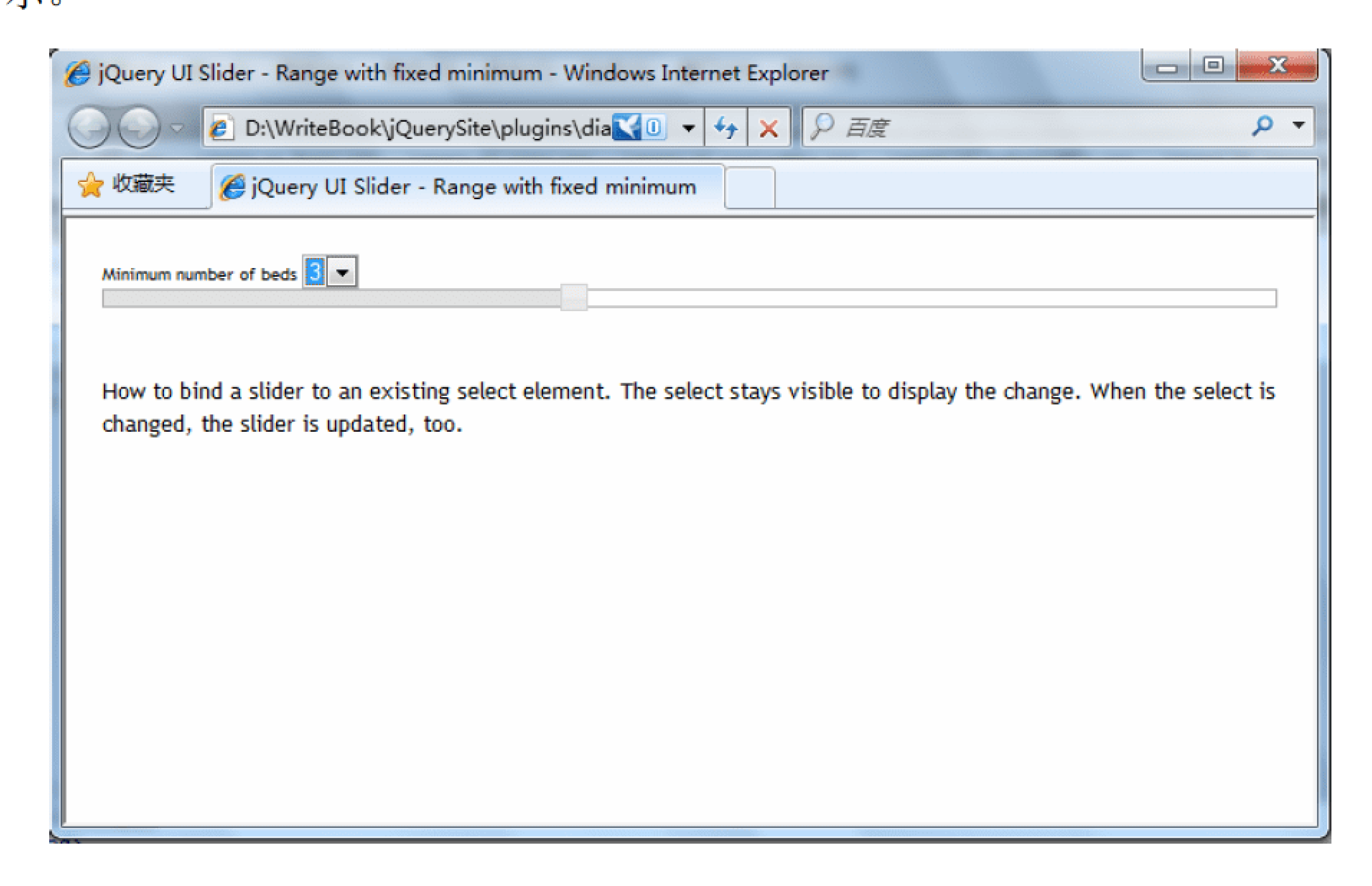


图 10.7　外部元素控制最大值变化滑动条

7. 垂直滑动条

这个示例使用滑动条的滑动方向和范围参数，并使用滑动条插件的初值及最大最小值属性和滑动事件。这里只是简单地将前面使用的滑动条纵向摆放，产生了垂直滑动条的效果。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
3       $( "#slider-vertical" ).slider({
4           orientation: "vertical",        //垂直滑动
5           range: "min",                   //最小表示范围
6           min: 0,                         //最小表示值为 0
7           max: 100,                       //最大表示值为 100
8           value: 60,                      //初始值为 60
9           slide: function( event, ui ) { //滑动条滑动事件
10              $( "#amount" ).val( ui.value );
11          }
12      });
13      $( "#amount" ).val( $( "#slider-vertical" ).slider( "value" ) );
14  });
15  </script>
```

上述代码第 4 行通过滑动条的滑动方向参数，设定了滑动条垂直滑动，并且最小值滑

动块固定在整个滑动条的下方。效果如图 10.8 所示。

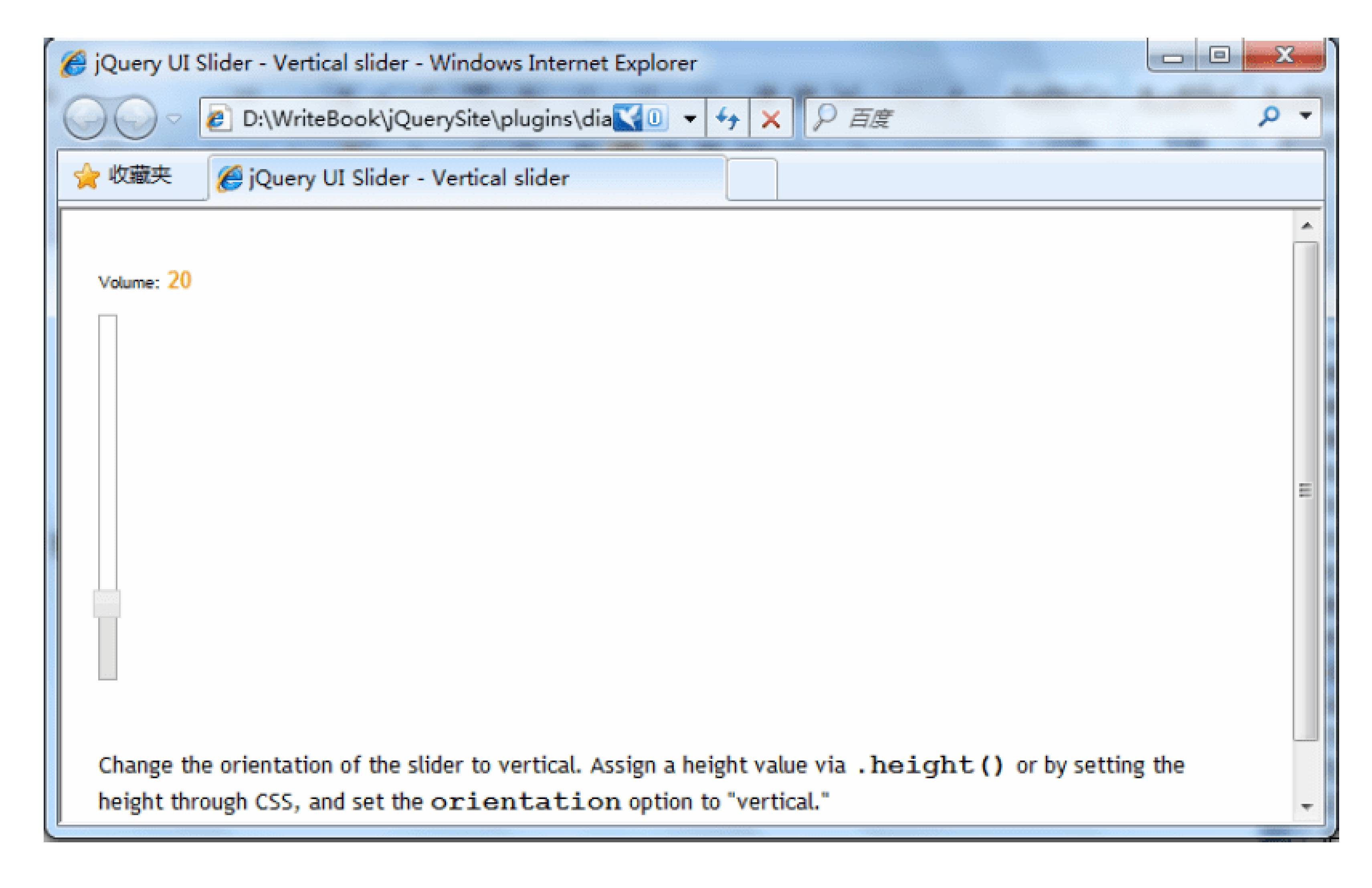


图 10.8　垂直滑动条

8．配色功能滑动条

这个示例实现通过表示三原色的三个滑动条的取值变化，产生不同颜色搭配。该示例使用滑动方向、范围、最大值、初始值属性及滑动块移动和值改变事件。通过三个滑动条中滑动块的移动代表三原色的不同的值，并对这些值进行运算，搭配出混合后的颜色。JavaScript 功能代码如下：

```
1   function hexFromRGB(r, g, b) {
2       var hex = [                         //三原色十六进制值字符串
3           r.toString( 16 ),
4           g.toString( 16 ),
5           b.toString( 16 )
6       ];
7       $.each( hex, function( nr, val ) { //每种颜色值用两位十六进制数表示
8           if ( val.length === 1 ) {
9               hex[ nr ] = "0" + val;
10          }
11      });
12      return hex.join( "" ).toUpperCase();//拼接成一个六位的十六进制字符串
13  }
14  function refreshSwatch() {
15      var red = $( "#red" ).slider( "value" ),
16          green = $( "#green" ).slider( "value" ),
17          blue = $( "#blue" ).slider( "value" ),
18          hex = hexFromRGB( red, green, blue );
19      $( "#swatch" ).css( "background-color", "#" + hex );
20  }
21  $(function() {
```

```
22      $( "#red, #green, #blue" ).slider({
23          orientation: "horizontal",      //水平滑动
24          range: "min",                   //最小表示范围功能
25          max: 255,                       //最大表示值
26          value: 127,                     //初始值
27          slide: refreshSwatch,           //滑动事件
28          change: refreshSwatch           //滑动块改变位置事件
29      });
30      $( "#red" ).slider( "value", 255 );
31      $( "#green" ).slider( "value", 140 );
32      $( "#blue" ).slider( "value", 60 );
33  });
34  </script>
```

上述代码第 1～13 行表示将三个不同滑动条的十进制整型值转换成一个六位的十六进制字符串。第 14～20 行表示提取三个滑动条的整型值并进行转换后将颜色显示出来。第 22～29 行是三个滑动条的初始化设定，并指定了两个事件所关联的函数。第 30～31 行分别指定三个滑动条的不同起始值。效果如图 10.9 所示。

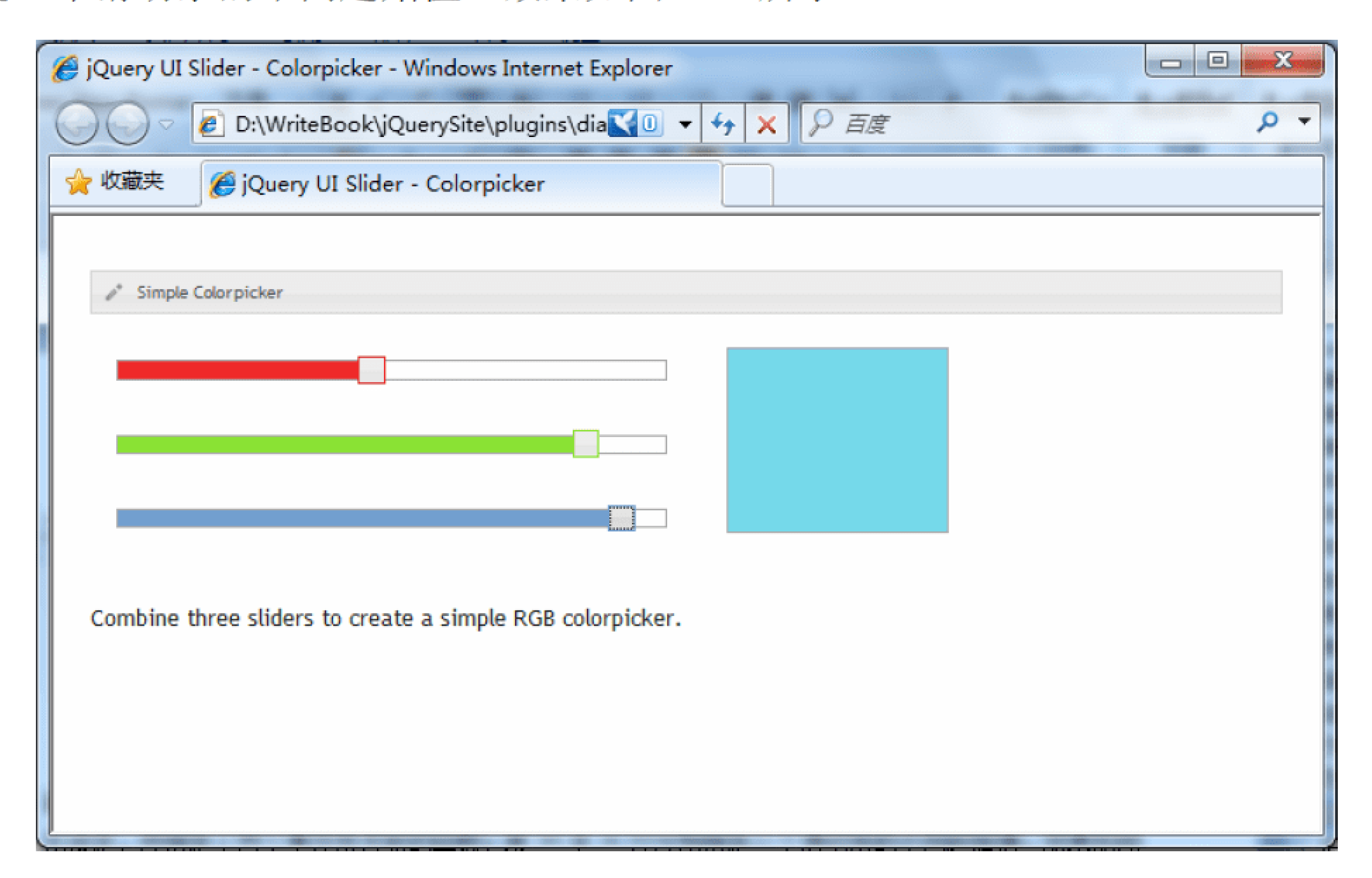


图 10.9　滑动条实现颜色搭配

9．内容滚动条

这个示例使用滑动条与内容层搭配产生滚动条效果，主要是对滑动条的初始值和滑动事件进行操作，并且在这个示例中滑动条可根据窗口大小自动调整宽度。与本章最开始讲解的例子类似，通过将滑动条的位移转换成内容的位移，就产生了内容滚动效果。JavaScript 功能代码如下：

```
1   <script>
2   $(function() {
```

```
3       //滚动区域
4       var scrollPane = $( ".scroll-pane" ),
5           scrollContent = $( ".scroll-content" );
6
7       //build slider
8       var scrollbar = $( ".scroll-bar" ).slider({
9           slide: function( event, ui ) {
10              if ( scrollContent.width() > scrollPane.width() ) {
                                    //判断显示内容宽度与显示区域宽度
11                  scrollContent.css( "margin-left", Math.round(
12                      ui.value / 100 * ( scrollPane.width() - scroll-
                        Content.width() )
13                  ) + "px" );     //设定内容显示位置
14              } else {
15                  scrollContent.css( "margin-left", 0 );
16              }
17          }
18      });
19
20      //
21      var handleHelper = scrollbar.find( ".ui-slider-handle" )
22      .mousedown(function() {
23          scrollbar.width( handleHelper.width() );
24      })
25      .mouseup(function() {
26          scrollbar.width( "100%" );
27      })  //设定滑动条的鼠标左键按下与释放事件
28      .append( "<span class='ui-icon ui-icon-grip-dotted-vertical'>
        </span>" )                      //添加图标
29      .wrap( "<div class='ui-handle-helper-parent'></div>" ).parent();
30
31      //change overflow to hidden now that slider handles the scrolling
32      scrollPane.css( "overflow", "hidden" );
33
34      //size scrollbar and handle proportionally to scroll distance
35      function sizeScrollbar() {
36          var remainder = scrollContent.width() - scrollPane.width();
            //内容宽度与显示区域宽度的差
37          var proportion = remainder / scrollContent.width();
            //未显示宽度占整个内容宽度的比例
38          var handleSize = scrollPane.width() - ( proportion * scrollPane.
            width() );
            //滑动区间的宽度
39          scrollbar.find( ".ui-slider-handle" ).css({
40              width: handleSize,
41              "margin-left": -handleSize / 2
42          });
43          handleHelper.width( "" ).width( scrollbar.width() - handleSize );
```

```
44      }
45
46      //reset slider value based on scroll content position
47      function resetValue() {
48          var remainder = scrollPane.width() - scrollContent.width();
49          var leftVal = scrollContent.css( "margin-left" ) === "auto" ? 0 :
50              parseInt( scrollContent.css( "margin-left" ) );
            //显示内容的左偏移量
51          var percentage = Math.round( leftVal / remainder * 100 );
            //左偏移量占整个未显示宽度的百分比
52          scrollbar.slider( "value", percentage );
53      }
54
55      //滑动块到最大位置,当窗口变大时,重新设置显示内容
56      function reflowContent() {
57              var showing = scrollContent.width() + parseInt(scroll-
                Content.css( "margin-left" ), 10 );
                //当前显示内容的偏移位置
58              var gap = scrollPane.width() - showing;
                //显示位置与显示区域的差
59              if ( gap > 0 ) {    //是否超出显示范围，如果大于 0，则可显示
60                  scrollContent.css( "margin-left", parseInt( scroll-
                    Content.css( "margin-left" ), 10 ) + gap );
61              }
62      }
63
64      //change handle position on window resize
65      $( window ).resize(function() {
66          resetValue();
67          sizeScrollbar();
68          reflowContent();
69      });
70      //init scrollbar size
71      setTimeout( sizeScrollbar, 10 );//safari wants a timeout
72  });
73  </script>
```

上述代码第 4、5 行获取显示内容的层对象。第 8～18 行建立滑动条，设定滑动事件。当内容长度大于显示区域宽度时，按照滑动块所在位置算出内容宽度百分比，进行内容的相对位移。第 21～24 行设定鼠标按下和抬起时滑动条宽度。第 28、29 行添加图标到滑动块，第 32 行设定浮动层隐藏。

第 35～44 行根据显示内容宽度与显示区域的宽度比来设定滑动块和滑动条的宽度。第 47～53 行根据显示内容的当前位置和宽度比来设定新的滑动块的位置。第 56～62 行设定显示内容的当前显示位置。第 65～69 行根据窗口大小的改变修改滑动块的位置。效果如图 10.10 所示。

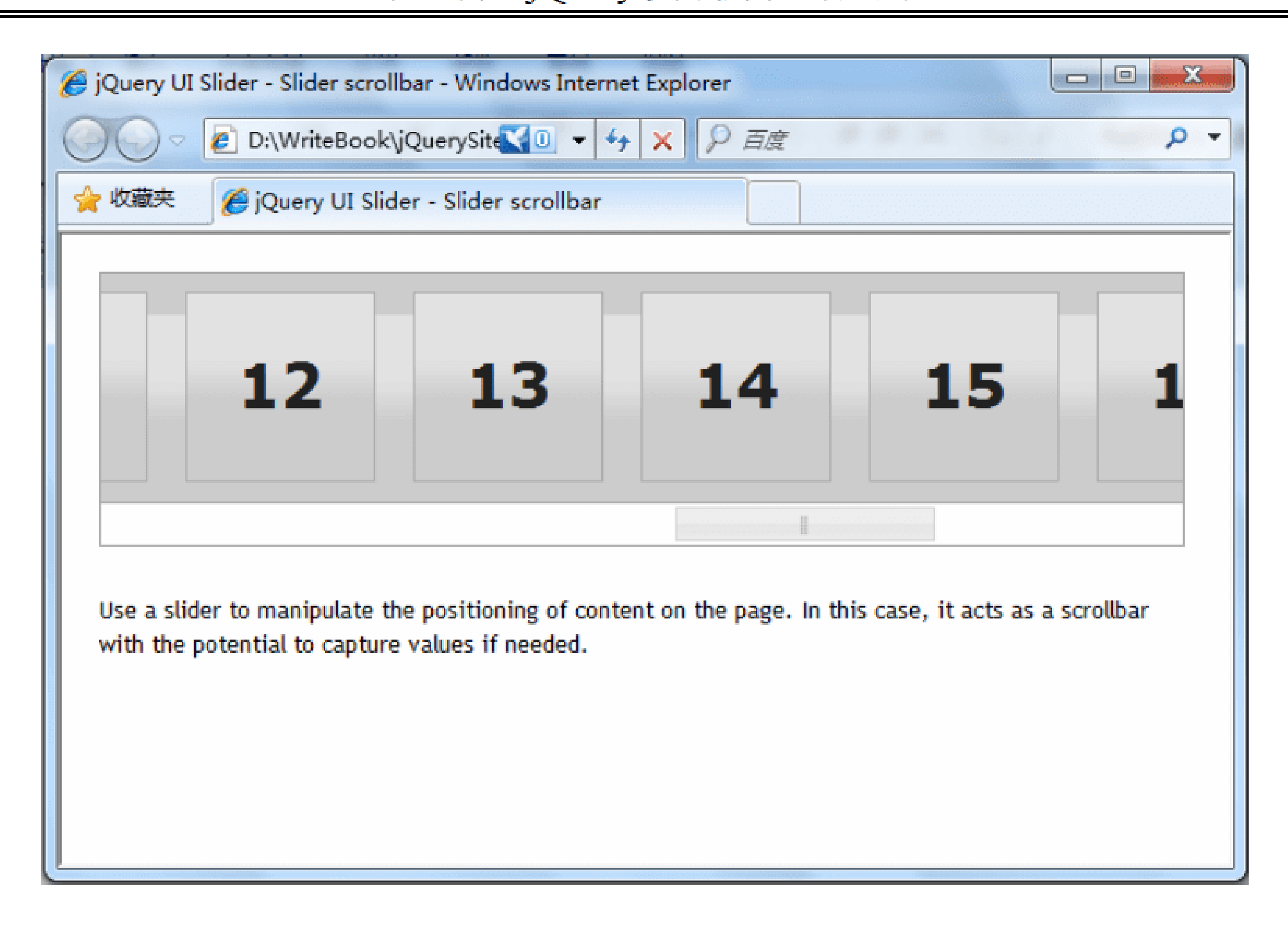


图 10.10　滑动条改变显示内容

10.2.2　jScrollPane 插件

jScrollPane 插件是一个跨浏览器的 jQuery 插件，它的英文网址为 http://jscrollpane.kelvinluck.com/。它可以将浏览器默认的滚动条转为 HTML 结构的，并可通过 CSS 灵活改变外观的滚动条形式。jScrollPane 灵活易用。

这个插件有一些相关属性，可以设置滚动条的样式和行为，如表 10.4 所示。

表 10.4　jScrollPane插件的属性说明

属　　性	类型	默认值	说　　明
showArrows	布尔	false	是否显示两端的箭头按钮
maintainPosition	布尔	true	当重新初始化时，滚动条是否保持原有位置
stickToBottom	布尔	false	在 maintainPosition 属性为真的时候，当整个滚动条已经到达底端的时候，保持这个状态
stickToRight	布尔	false	在 maintainPosition 属性为真的时候，当整个滚动条已经到达右端的时候，保持这个状态
autoReinitialise	布尔	false	在第一次初始化后，是否再次自动初始化。当内容发生改变时，不建议使用
autoReinitialiseDelay	整型	500	自动初始化间隔的毫秒数
verticalDragMinHeight	整型	0	垂直拖动的最小高度
verticalDragMaxHeight	整型	99999	垂直拖动的最大高度
horizontalDragMinWidth	整型	0	水平拖动的最小宽度
horizontalDragMaxWidth	整型	99999	水平拖动的最大宽度
contentWidth	整型		滚动条内容宽度
animateScroll	布尔	false	当调用滚动方法时使用动画效果
animateDuration	整型	300	动画滚动的持续时间，单位为毫秒
animateEase	字符串	'linear'	动画类型

续表

属　　性	类型	默认值	说　　明
hijackInternalLinks	布尔	false	是否在插件作用范围内使用书签链接
verticalGutter	整型	4	设置显示内容到垂直滚动条边框空格的个数
horizontalGutter	整型	4	设置显示内容到水平滚动条边框空格的个数
mouseWheelSpeed	整型	10	鼠标滚轮步进的位移量
arrowButtonSpeed	整型	10	单击箭头按钮移动的单位位移量
arrowRepeatFreq	整型	100	箭头按钮点击频率，单位为毫秒
arrowScrollOnHover	布尔	false	当鼠标悬停在箭头按钮上时，是否触发滚动
verticalArrowPositions	字符串	split	垂直滚动条箭头按钮出现的位置，可选值为 split|before|after|os
horizontalArrowPositions	字符串	splite	水平滚动条箭头按钮出现的位置，可选值为 split|before|after|os
enableKeyboardNavigation	布尔	true	是否可以使用键盘方向键操作滚动条
hideFocus	布尔	false	是否隐藏焦点，建议设为隐藏
clickOnTrack	布尔	true	是否允许单击改变滚动条滚动位置
trackClickSpeed	整型	30	鼠标单击一次的步进值
trackClickRepeatFreq	整型	100	鼠标单击间隔时间，单位为毫秒

滚动条有一些函数可以控制滚动条的相关操作，如表 10.5 所示。

表 10.5　jScrollPane插件的函数说明

函 数 形 式	说　　明
reinitialise(s)	再次初始化滚动条，如果有新的属性设置则重新配置，如果没有则沿用上一次初始化的属性设置
scrollToElement(ele, stickToTop, animate)	将一个 jQuery 对象、DOM 节点或者 jQuery 选择器字符串所指向的元素移动到指定显示位置，如果 stickToTop 为真，则移动到可见位置的最上端，否则在可见位置下端，也可指定动画效果
scrollTo (destX, destY, animate)	设定滚动条滚动到内容的指定纵向横向位置，可以添加动画效果
scrollToX (destX, animate)	设定滚动条横向滚动到的位置，可以添加动画效果
scrollToY (destY, animate)	设定滚动条纵向滚动到的位置，可以添加动画效果
scrollToPercentX (destPercentX, animate)	设定滚动条横向滚动到占最大值指定百分比的位置，可以添加动画效果
scrollToPercentY (destPercentY, animate)	设定滚动条纵向滚动到占最大值指定百分比的位置，可以添加动画效果
scrollBy(deltaX, deltaY, animate)	设定横向纵向滚动的相对偏移量，单位为像素，可以添加动画效果
scrollByX(deltaX, animate)	设定横向滚动的相对偏移量，单位为像素，可以添加动画效果
scrollByY(deltaY, animate)	设定纵向滚动的相对偏移量，单位为像素，可以添加动画效果
positionDragX (x, animate)	水平拖动的位置，可以添加动画效果

续表

函 数 形 式	说　　明
positionDragY (y, animate)	垂直拖动的位置，可以添加动画效果
animate (ele, prop, value, stepCallback)	当滚动条动画到一个新位置时调用这个函数，ele 为使用动画的元素，prop 为使用动画的属性，value 为动画中的属性值，stepCallback 为更新属性值需要执行的方法
getContentPositionX()	返回滚动条的当前横向 X 轴位置
getContentPositionY()	返回滚动条的当前纵向 Y 轴位置
getContentWidth()	返回滚动条宽度
getContentHeight()	返回滚动条高度
getIsScrollableH()	返回判断是否有横向滚动条
getPercentScrolledX()	返回滚动条相对显示区域横向位置
getPercentScrolledY()	返回滚动条相对显示区域纵向位置
getIsScrollableV()	返回判断是否有纵向滚动条
getContentPane()	获得一个滚动条的引用
scrollToBottom(animate)	滚动条滚动到下端
hijackInternalLinks()	书签链接操作
destroy()	销毁滚动条

下面通过示例来介绍一下这个插件的使用方法。

1. 默认样式滚动条

这个示例使用滚动条的默认参数形式，只调用了滚动条插件的初始化函数，并适当加入了一些简单属性设置。其中，HTML 代码和 CSS 代码请参考光盘内容。JavaScript 功能代码如下：

```
1        <script type="text/JavaScript">
2            $(function()
3            {
4                //设置默认属性选择值，显示上下移动箭头
5                $.extend($.fn.jScrollPane.defaults, {showArrows:true});
6                //上一行代码也可进行如下设定
7                //$.fn.jScrollPane.defaults.showArrows = true;
8                //插件初始化
9                $('#pane1').jScrollPane();
10               $('#pane2').jScrollPane();
11               //将第 3 个插件的上下移动箭头取消，并设定滚动条的宽和间隔距离
12               $('#pane3').jScrollPane({scrollbarWidth:20, scrollbar-
                 Margin:10, showArrows:false});
13               $('#pane4').jScrollPane();
14
15               //向第 4 个插件中加入内容事件
16               //当向其中加入内容时，会要求重新初始化这个插件
17               $('#add-content').bind(
18                   'click',
19                   function()
```

```
20                  {
21                      $('#pane4').append($('<p></p>').html($('#intro').
                        html())).jScrollPane();
22                  }
23              );
24              //从第 4 个插件中删除内容事件
25              //这里也会要求重新初始化插件
26              $('#remove-content').bind(
27                  'click',
28                  function()
29                  {
30                      $('#pane4').empty().append($('<p></p>').html
                        ($('#intro').html())).jScrollPane();
31                  }
32              );
33          });
34
35      </script>
```

上述代码第 5 行设定了插件的显示箭头按钮的属性。第 9、10、12、13 行初始化了 4 个滚动条。第 12 行配置了滚动条的宽为 20 像素，间隔为 10 像素，不显示箭头按钮。第 17～23 行绑定了向第 4 个滚动条所在区域添加内容的事件。第 26～32 行绑定了从第 4 个滚动条所在区域删除内容的事件。效果如图 10.11 和图 10.12 所示。

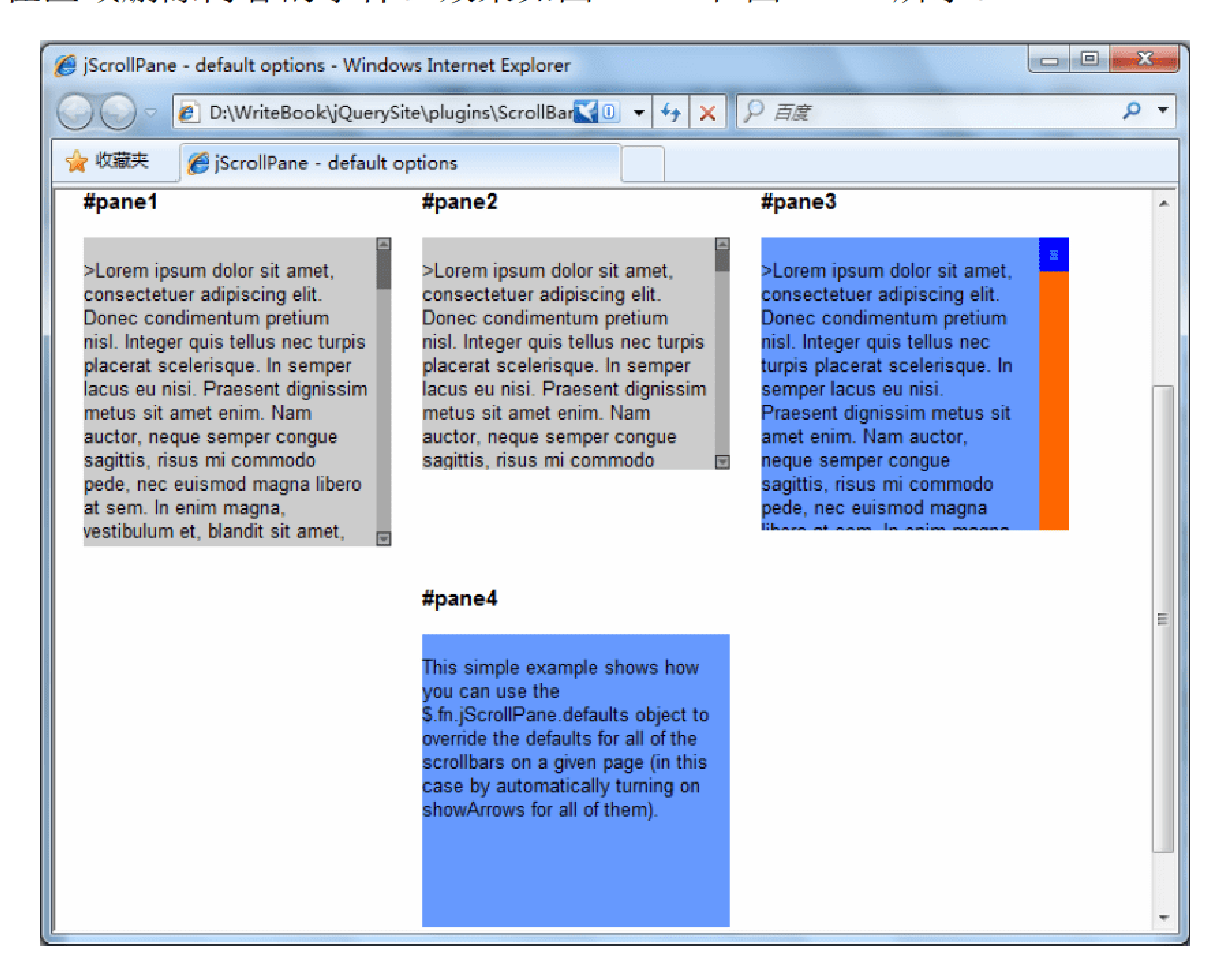


图 10.11　通过默认选项创建滚动条

2. 自定义浏览器滚动条

这个示例利用 jScrollPane 创建的滚动条来代替浏览器默认的滚动条。在某些情况下浏

览器自身的滚动条的样式满足不了需要，此时可以使用这个插件创建属于我们自己的浏览器滚动条。JavaScript 功能代码如下：

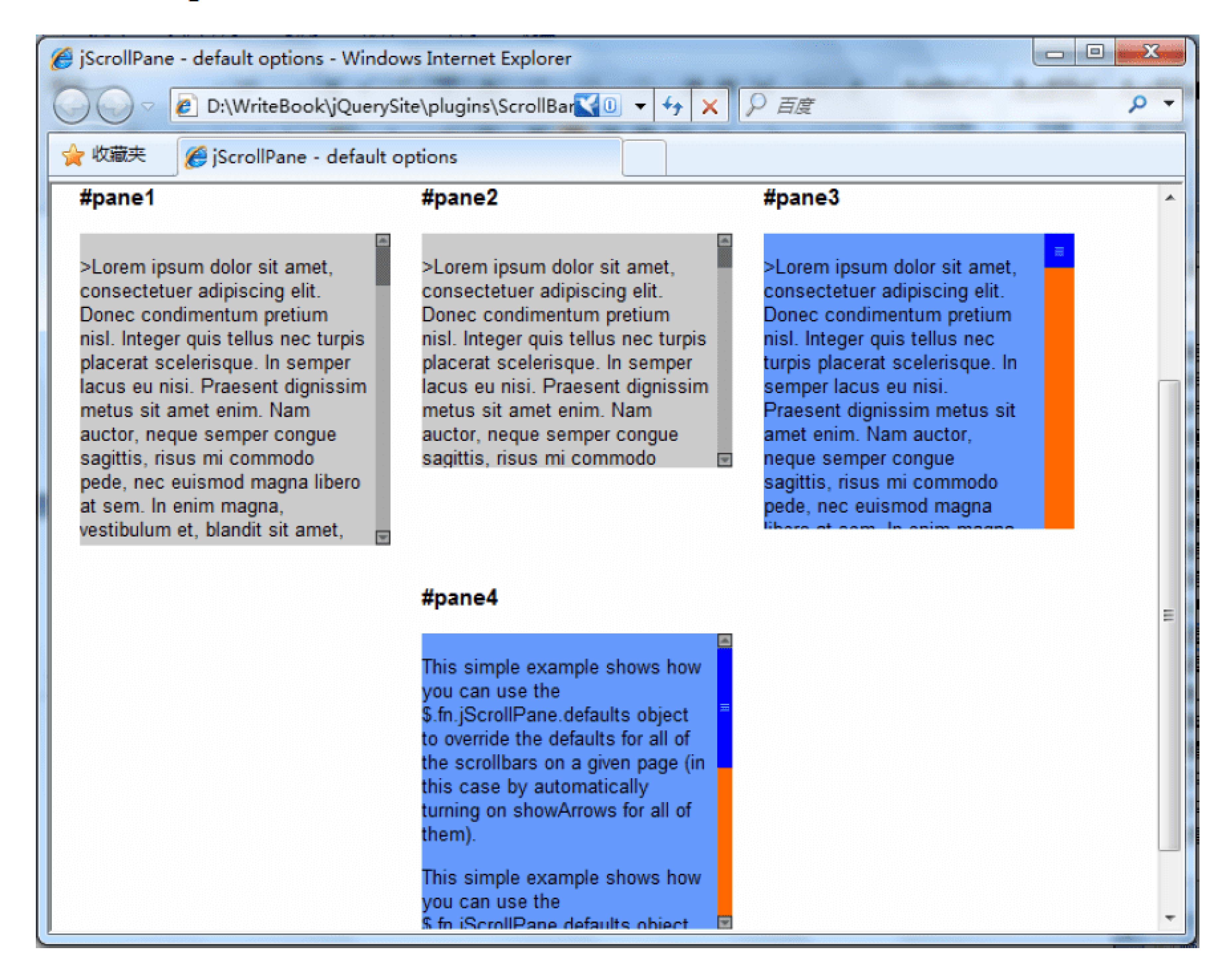


图 10.12　向滚动条所在区域动态添加内容后效果

```
1        <script type="text/JavaScript">
2            $(function()
3            {
4                $.extend($.fn.jScrollPane.defaults, {
5                    showArrows:true,        //显示表示滑动方向的箭头
6                    scrollbarWidth:15,      //设定滑动条宽度
7                    arrowSize:16            //设置箭头图标大小
8                    }
9                );
10               var isResizing;
11               //获取整个容器高度
12               var setContainerHeight = function()
13               {
14                   //IE 触发浏览器大小改变事件
15                   //避免浏览器冲突
16                   if (!isResizing) {
17                       isResizing = true;
18                       $w = $(window);
19                       $c = $('#container');
20                       var p = (parseInt($c.css('paddingLeft')) || 0) +
                         (parseInt($c.css('paddingRight')) || 0);
21                       $('body>.jScrollPaneContainer').css({'height':
                         $w.height() + 'px', 'width': $w.width() + 'px'});
22                       $c.css({'height': ($w.height()-p) + 'px', 'width':
                         ($w.width() - p) + 'px', 'overflow':'auto'});
```

```
23                        $c.jScrollPane();
24                        isResizing = false;
25                    }
26                }
27                $(window).bind('resize', setContainerHeight);
28                setContainerHeight();
29
30                //这里需要再次调用获取容器高度函数
31                setContainerHeight();
32            });
33        </script>
```

上述代码第 4～9 行对滚动条设置宽度为 15 像素，滚动条使用箭头按钮，按钮大小为 16 像素。第 12～26 行定义了对浏览器添加自定义滚动条函数。其中第 17 行设定浏览器窗口可调整尺寸标志。第 18 行获取浏览器窗口对象。第 19 行获取最底层容器层对象。第 20 行获取文字内容与边框之间的距离。第 21 行设置滚动条的尺寸。第 22 行设置容器层的尺寸。第 23 行初始化滚动条。第 27 行绑定窗口的改变大小事件。最后调用添加自定义滚动条函数，文档中说明这个函数要调用两次。效果如图 10.13 所示。

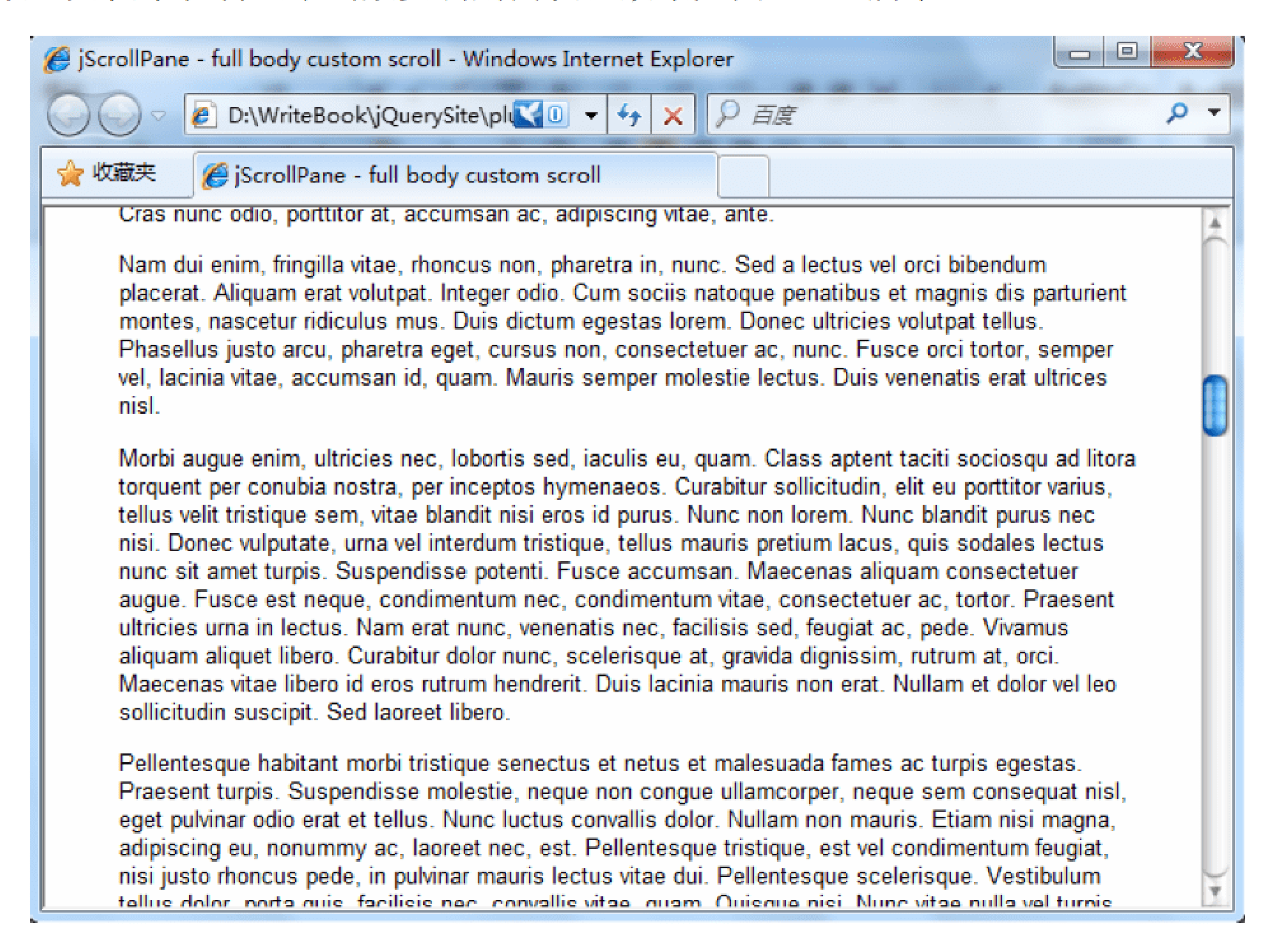


图 10.13　自定义浏览器窗口滚动条

3．重新初始化属性的应用

这个示例使用了插件的重新初始化属性，要实现当图片加载完成后再次初始化操作。当图片或者内容大小较大时，加载时间会有延迟，因此当图片全部加载完成或者内容全部加载后，需要根据实际内容的大小重新初始化滚动条的大小及滚动范围。JavaScript 功能代码如下：

```
1        <script type="text/JavaScript">
2            $(function()
3            {
4                //一旦图片被完全加载，则重新初始化插件
5                $('#pane1')
6                    .jScrollPane(
7                        {
8                            showArrows:true,    //显示表示滑动方向箭头
9                            scrollbarWidth: 20, //滑动条宽度
10                           reinitialiseOnImageLoad: true
                                                 //图片加载后重新初始化插件
11                       }
12                   );
13           });
14       </script>
```

上述代码第 10 行就是再次初始化选项，在这里设置为真则表示允许滚动条再次初始化。效果如图 10.14 所示。

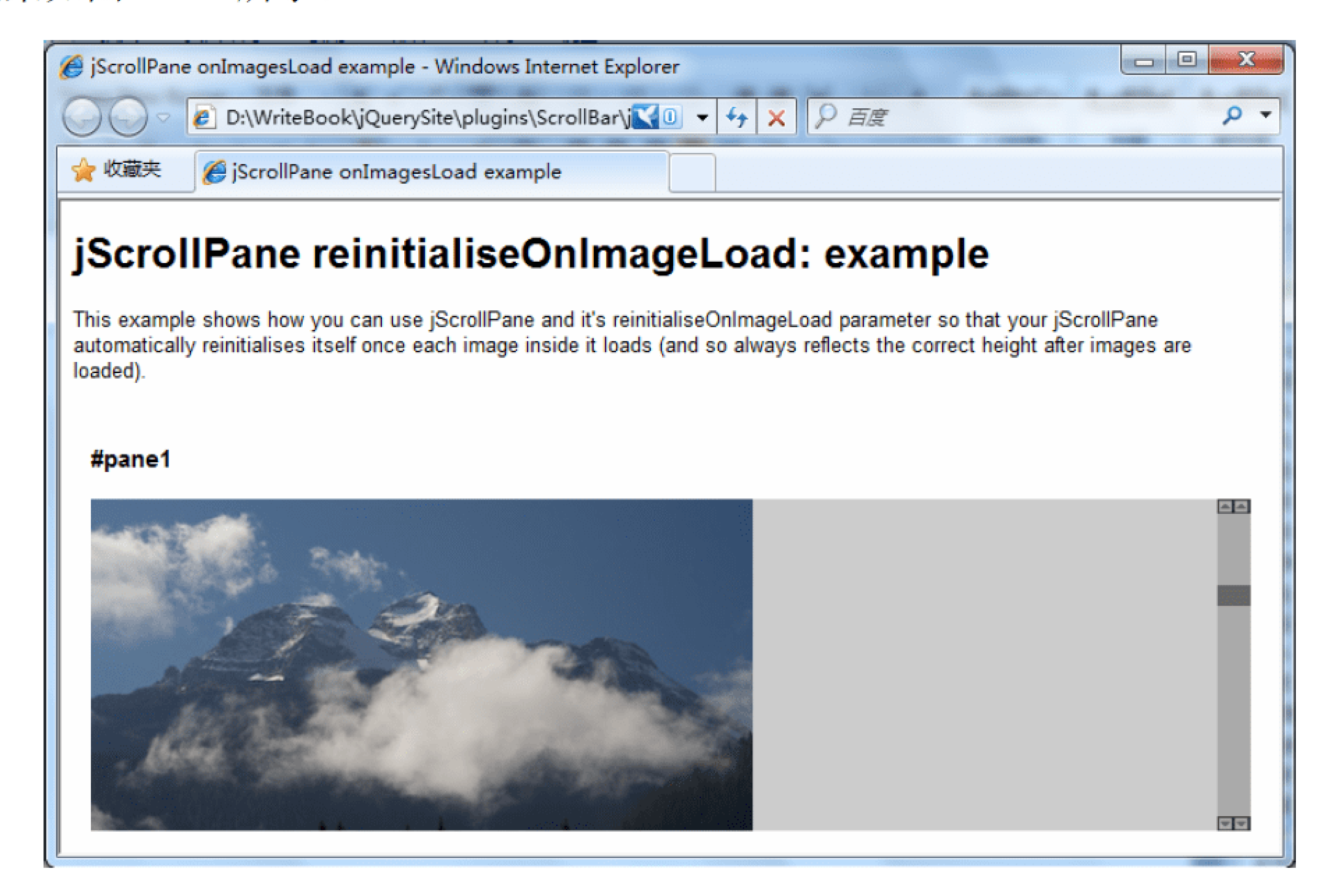


图 10.14　再次加载滚动条

4．滚动事件举例

这个示例使用了滚动条的滚动事件，并在这个事件中添加了记录浏览器日志的操作。JavaScript 功能代码如下：

```
1        <script type="text/JavaScript">
2            $(function()
3            {
4                //简单初始化插件
5                $('#pane1').jScrollPane();
6                $('.scroll-pane').bind(    //绑定滑动事件
```

```
7                       'scroll',
8                       function(event)
9                       {
10                          console.log(event.target);
11                      }
12                  )
13              });
14          </script>
```

上述代码第 6～12 行绑定了滚动条的滚动事件。第 7 行指定绑定事件。第 10 行表示在滚动事件发生时记录浏览器日志。效果如图 10.15（此图效果是在 IE 8 中实现的）所示。

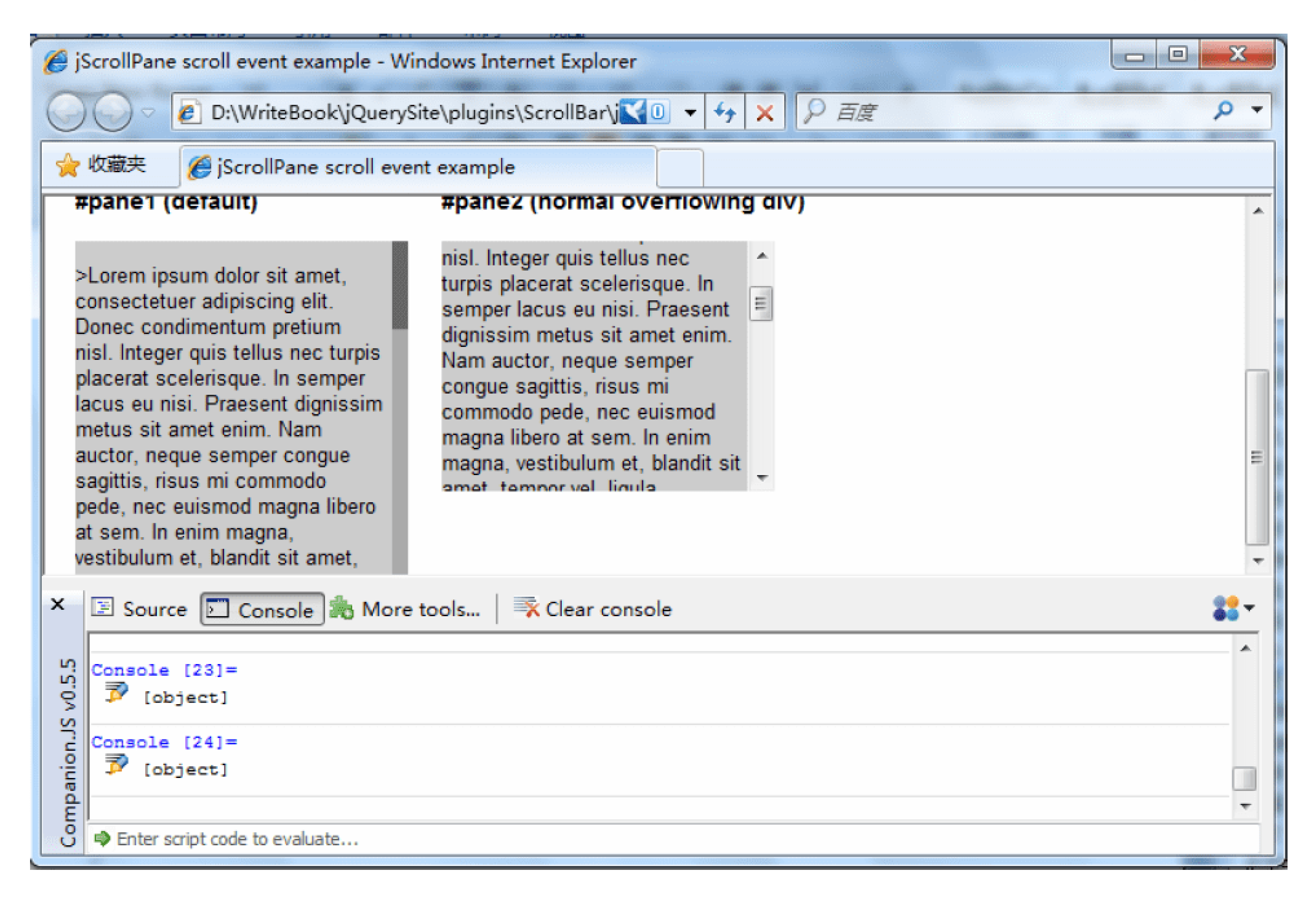


图 10.15　利用滚动条滚动事件记录浏览器日志

5．滚动条编程设定滚动位置

这个示例使用了滚动条更改滚动位置的相关操作函数，通过编程动态修改滚动条位置。也可以通过外界元素的变化影响滚动条的变化，更改滚动块的位置。JavaScript 功能代码如下：

```
1       <script type="text/JavaScript">
2           $(function()
3           {
4               //分别初始化三个插件
5               $('#pane1').jScrollPane();
6               $('#pane2').jScrollPane({showArrows:true});
7               $('#pane3').jScrollPane({scrollbarWidth:20, scrollbar-
                Margin:10, animateTo:true});
8             //初始化页面其他链接的单击事件，修改滚动块的位置
9             $('.scroll-to-demo').bind(
```

```
10                  'click',
11                  function()
12                  {
13                      $('#pane1')[0].scrollTo(200);
14                      return false;
15                  }
16              );
17              $('.scroll-by-demo').bind(
18                  'click',
19                  function()
20                  {
21                      $('#pane2')[0].scrollBy(parseInt($(this).
                        attr('rel')));
22                      return false;
23                  }
24              );
25              var $targets = $('#pane3-scroll-targets');
26              var $pane3 = $('#pane3');
27              var pane3top = parseInt($pane3.offset().top);
28              $('#pane3 p').each(
29                  function(index)
30                  {
31                      $targets.append(
32                          $('<li></li>').append(
33                              $('<a></a>')
34                                  .attr({'href':'JavaScript:;', 'rel':
                                    $(this).offset().top})
35                                  .text('Scroll to paragraph ' + (index+1))
36                                  .bind(
37                                      'click',
38                                      function()
39                                      {
40                                          $pane3[0].scrollTo(parseInt
                                            ($(this).attr('rel')) - pane3top);
41                                      }
42                                  )
43                              )
44                      );
45                  }
46              );
47              //对第 4 个插件设定滚动块的编程改变位置事件
48              var $pane4 = $('#pane4');
49              $pane4.jScrollPane({animateTo:true});
50              $('a.scroll-to-element-demo').bind(
51                  'click',
52                  function()
53                  {
54                      var targetElementSelectorString = $(this).
                        attr('rel');
55                      $pane4[0].scrollTo(targetElementSelectorString);
                                    //设定滑动到指定位置
56                      return false;
```

```
57                    }
58                );
59            });
60        </script>
```

上述代码第 5、6、7 行和第 49 行分别初始化了 4 个滚动条，其中第 3 个滚动条设定了滚动条宽度为 20 像素，边框间隔为 10 像素，使用动画。第 4 个滚动条也使用了动画。第 9～16 行设定第 1 个滚动条动态滚动事件，将滚动块定位在 200 位置。第 17～24 行设定了第 2 个滚动条动态滚动事件，利用链接元素的单击事件改变滚动块位置，每次都以链接的 rel 属性值为移动量。

第 25～46 行定义了对第 3 个滚动条的滚动触发事件，并以每一段落的起始行位置为移动位移量。第 48～58 行定义了第 4 个滚动条的滚动触发事件，移动的位置是通过 jQuery 选择器而不是通过坐标位置定位的。效果如图 10.16 和图 10.17 所示。

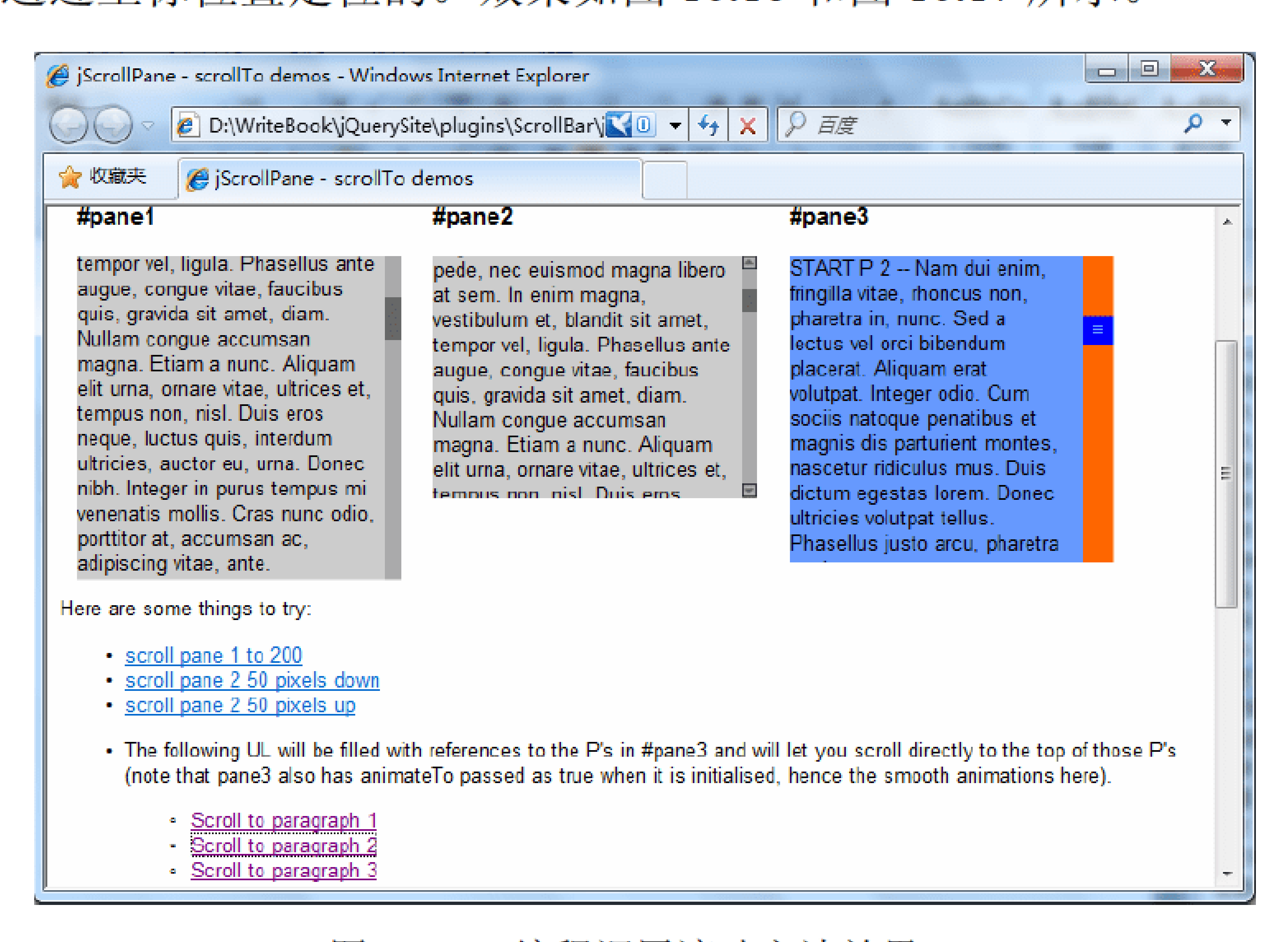


图 10.16　编程调用滚动方法效果一

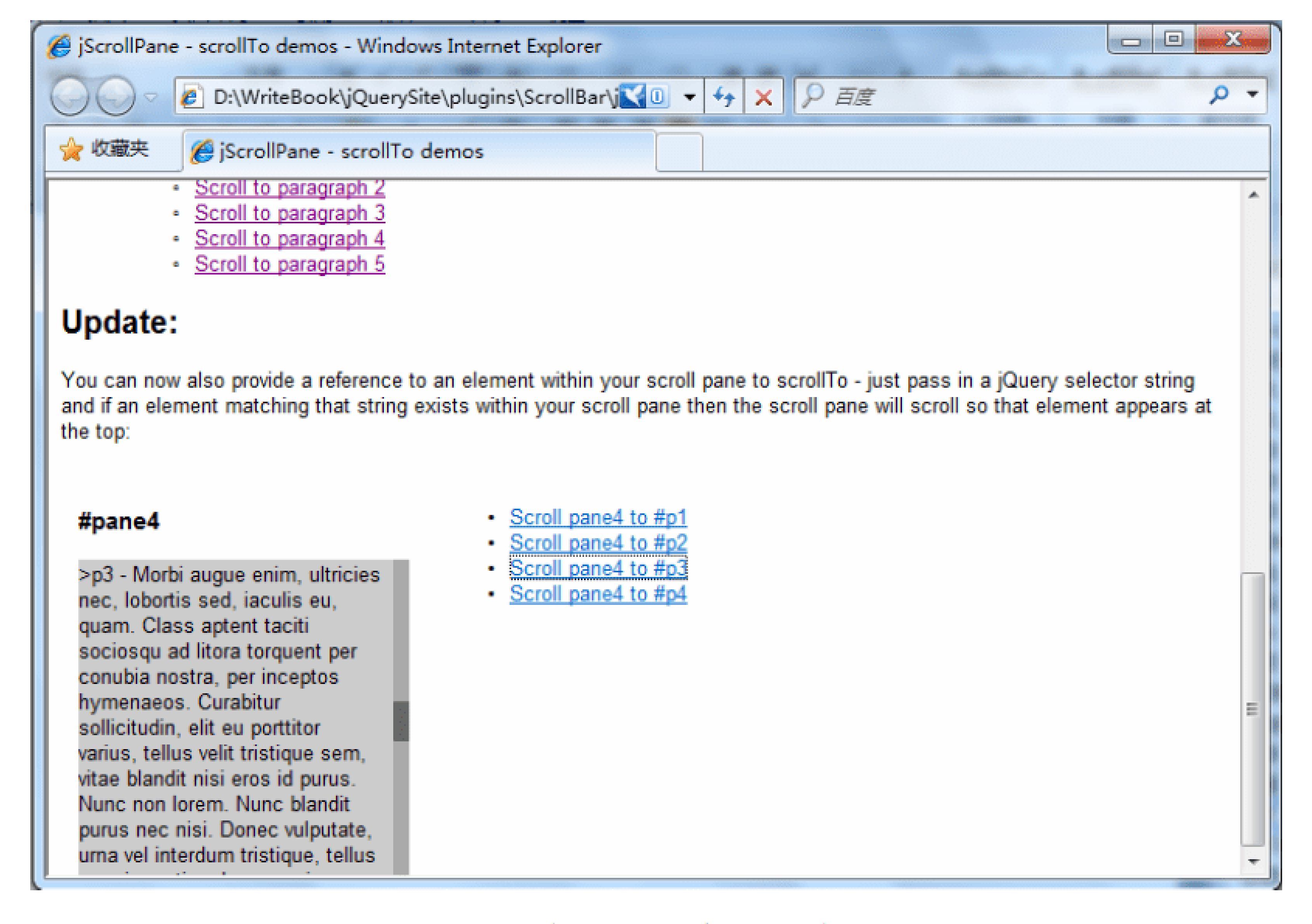


图 10.17　编程调用滚动方法效果二

6．加入动画效果的滚动

这个示例通过设定不同的超链接的单击事件，使对应的滚动条以不同的动画形式滚动定位到不同位置。滚动条动画效果的移动也是一种全新的用户体验。JavaScript 功能代码如下：

```
1       <script type="text/JavaScript">
2           $(function()
3           {
                //初始化各个插件
                //设定动画间隔时间，以及动画移动时的单位移动距离
4               $('#pane1').jScrollPane({animateTo:true, animateInterval:
                50, animateStep:5});
5               $('#pane2').jScrollPane({animateTo:true, animateInterval:
                50, animateStep:3});
6               $('#pane3').jScrollPane({animateTo:true, animateInterval:
                150, animateStep:5});
7               $('#pane4').jScrollPane({animateTo:true, animateInterval:
                150, animateStep:3});
8               $('#pane5').jScrollPane({animateTo:false});
                //超链接单击事件，触发滚动条的动画移动效果
9               $('a.scroll-to-element-demo').bind(
10                  'click',
11                  function()
12                  {
13                      $this = $(this);
14                      var destinationSelector = $(this).attr('rel');//
15                      $('.scroll-pane', $this.parent().parent().
                        parent()).each(
16                          function()
17                          {
18                              this.scrollTo(destinationSelector);
                                    //设定滑动到指定位置
19                          }
20                      );
21                      return false;
22                  }
23              );
24
25          });
26      </script>
```

上述代码第 4～8 行初始化了 5 个滚动条，它们具有不同的动画属性。第 9～23 行对每个滚动条都使用了 4 个超链接，利用超链接的单击事件触发滚动条滚动，并定位到每个段落的起始位置。静态效果如图 10.18 所示。

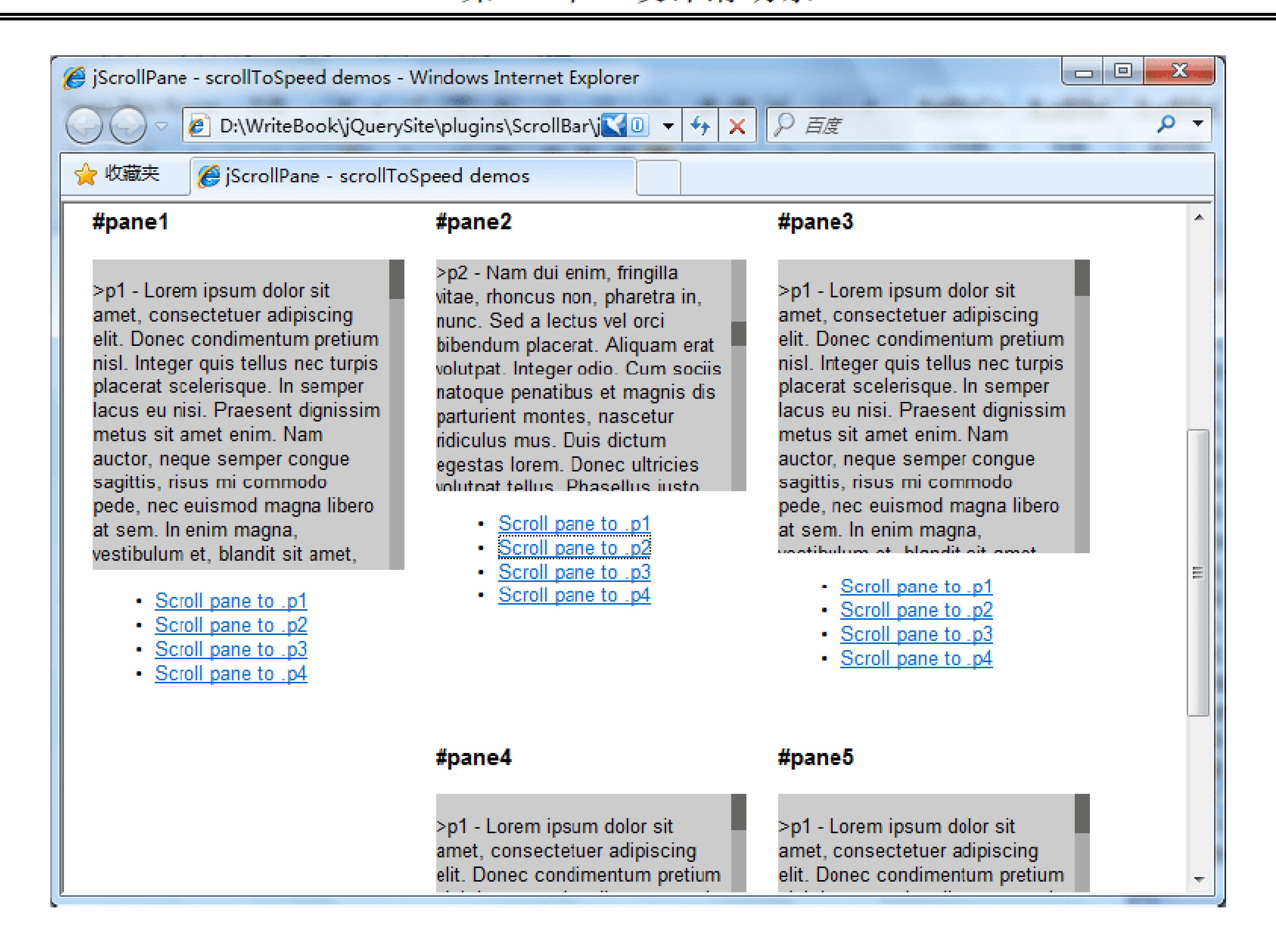


图 10.18 设置不同动画效果的滚动条滚动静态效果

10.3 小　　结

本章对如何用 jQuery 创建滑动条进行了讲解。主要内容包括基本滑动条实现原理、滑动条的事件过程处理，并介绍了两个滑动条插件。重点部分是滑动条的实现原理及滑动条插件的使用方法。滑动条的实现原理是本章的难点。下一章将讲解 jQuery 页面编辑器插件。

10.4 习　　题

【习题 1】利用本章所讲解内容自定义实现横向滑动条。

【习题 2】练习 jScrollPane 插件的使用方法。

第 11 章　页面编辑器插件

页面编辑器是在浏览器中创建一个具有文本编辑功能、类似文本编辑软件的区域。这种页面效果可以应用于网站论坛、网站后台管理、B/S 结构的 OA 平台等 CMS。它方便用户在浏览器中编辑文章，并在编辑完成后直接发表。本章将介绍 6 个页面编辑器插件。

11.1　markItUp 插件

markItUp 插件是一个轻量级的、具有客户定制并富有弹性的编辑引擎、面向开发人员的 jQuery 插件。它可应用于 CMS（内容管理系统）、博客、网站论坛等。这个插件的英文网址为 http://markitup.jaysalvat.com/home/，可以在该网址下载插件文件、文档及相关示例。

该插件的特点如下。

（1）可快捷地整合进新的项目中。

（2）支持键盘快捷键。

（3）具有工具栏和下拉菜单。

（4）灵活的客户定制。

（5）使用的随意性。

（6）Ajax 动态预览。

（7）可定制主题。

下面就来介绍这个插件的使用方法。

11.1.1　安装插件

在安装过程中，在引入插件文件之前需要加载 jQuery 库文件，并在引入插件文件之后继续引用插件的配置文件，如下所示。

（1）引入 JS 文件：

```
<!-- jQuery -->
<script type="text/JavaScript" src="../../../jslib/jquery-1.6.js"></script>
<!-- markItUp! -->
<script  type="text/JavaScript"  src="markitup/jquery.markitup.js"></script>
                                    //引入插件文件
```

（2）引入配置插件的 JSON 文件：

```
<!-- markItUp! toolbar settings -->
<script type="text/JavaScript" src="markitup/sets/default/set.js"></script>
                                            //引入插件要使用的 JSON 文件
```

（3）引入 CSS 样式文件：

```
<!-- markItUp! skin -->
<link rel="stylesheet" type="text/css" href="markitup/skins/markitup/
style.css" />
<!--  markItUp! toolbar skin -->
<link rel="stylesheet" type="text/css" href="markitup/sets/default/style.
css" />
```

（4）加入插件调用 JavaScript 功能代码。在调用插件初始化方法时，可以使用元素 id、元素的 class、元素标记名：

```
<script type="text/JavaScript">
<!--
    $('#markItUp').markItUp(mySettings);    //初始化插件
-->
</script>
```

11.1.2　键盘的使用

键盘的使用主要有以下几种。

（1）Ctrl+其他键，标记插入。

（2）Shift+其他键，新标记的插入。

（3）Ctrl+Shift+其他键，在选中的所有行添加标记。

（4）Alt+其他键，使用可选标记插入。

其中，“其他键”为除去 Ctrl、Shift、Alt 等键的按键，主要是标记使用的键盘快捷操作。例如，当需要在编辑器内添加一对粗体字标记时，可以使用 Ctrl+B 组合键来操作。

11.1.3　标记语言定制

标记语言定制主要设定插件可以处理的是何种标记语言、这种标记语言的语法如何，以及用户在编辑器中插入该标记语言时的行为特征等。标记语言定制包含若干属性，如表 11.1 所示。

表 11.1　标记语言定制的属性说明

属　　性	类　　型	说　　明
nameSpace	字符串	为封装的 DIV 指定一个类名，避免 CSS 样式冲突
resizeHandle	布尔	是否允许调整编辑器大小
previewAutoRefresh	布尔	当编辑内容改变时，是否刷新预览窗口
previewParserPath	字符串	自定义的预览语法分析
previewParserVar	字符串	发送给语法分析的内容名称
previewTemplatePath	字符串	预览模板位置

续表

属　　性	类　　型	说　　明
previewParser	函数	预览前分析内容的语法函数
previewPosition	字符串	预览窗口的位置
onEnter		定义当回车键按下时的操作
onCtrlEnter		定义当 Ctrl 和回车键同时按下时的操作
onShiftEnter		定义当 Shift 和回车键同时按下时的操作
onTab		定义当 Tab 键按下时的操作
beforeInsert	函数	定义在插入标记前执行的函数
afterInsert	函数	定义在插入标记后执行的函数
markupSet		标记集合操作为编辑器设置属性

markupSet 中的属性说明如表 11.2 所示。

表 11.2　markupSet中的属性说明

属　　性	类　　型	说　　明
name	字符串	按钮名称
classname	字符串	按钮应用的类名
key	字符串	按钮对应的快捷键
openWith	字符串或函数	在选择位置前标记添加
closeWith	字符串或函数	在选择位置后标记添加
replaceWith	字符串或函数	在鼠标或选择位置替换文本
openBlockWith	字符串或函数	在完整的块前加入文本
closeBlockWith	字符串或函数	在完整的块后加入文本
mutiline	布尔	标记是否插入每一行或者一个完整的块
placeHolder	字符串或函数	占位符文本被插入
beforeInsert	函数	标记插入前被调用的函数
afterInsert	函数	标记插入后被调用的函数
beforeMutiInsert	函数	多行标记插入前调用的函数
afterMutiInsert	函数	多行标记插入后调用的函数
dropMenu		打开下拉菜单的按钮

下面用一个示例来介绍这个插件的使用方法。这个示例主要是想介绍初始化插件时如何通过配置文件配置插件样式及功能。在这个示例中使用了插件的开始标记、结束标记、填充文本内容等行为属性，以及如何通过插件内置删除函数在页面中删除插件效果。下面看一下 JavaScript 的功能代码：

```
1    <script type="text/JavaScript">
     <!--
2    $(document).ready(function()    {
     //添加插件及应用配置信息
     //$('textarea').markItUp( { Settings }, { OptionalExtraSettings } );
3        $('#markItUp').markItUp(mySettings);
```

```
    //可以在任何位置添加任意内容
    //$.markItUp( { Settings } );
4       $('.add').click(function() {
5           $.markItUp( {   openWith:'<opening tag>',
6                       closeWith:'<closing tag>',
7                       placeHolder:"New content"
8                   }
9               );
10      return false;
11      });

    //在任意时刻添加或者删除插件效果
    //$(textarea).markItUpRemove();
12      $('.toggle').click(function() {
13          if ($("#markItUp.markItUpEditor").length === 1) {
14              $("#markItUp").markItUpRemove();
15              $("span", this).text("get markItUp! back");
16          } else {
17              $('#markItUp').markItUp(mySettings);
18              $("span", this).text("remove markItUp!");
19          }
20          return false;
21      });
22  });
    -->
23  </script>
```

上述代码第 3 行初始化了这个页面编辑器，其中利用了 set.js 文件中的设置内容。第 4～11 行是向编辑器中添加带有标记的文本。第 5 行进行编辑器的参数设定，设定文本内容的起始标记。第 6 行设定文本内容的结束标记。第 7 行为文本占位符；第 12～21 行，设定对编辑器的添加与删除操作。第 13 行判定页面中是否已经存在编辑器。第 14、15 行表示如果存在编辑器则删除，并修改单击元素的文本。第 17、18 行表示如果页面中不存在编辑器则创建，同时修改单击元素的文本。

为了对上面讲解的编辑器的相关属性有所了解，下面来看一下如何对编辑器进行常用配置，以 set.js 文件为例。

```
1 var mySettings = {
2   onShiftEnter:   {keepDefault:false, replaceWith:'<br />\n'},
3   onCtrlEnter:    {keepDefault:false, openWith:'\n<p>', closeWith:
    '</p>'},
4   onTab:          {keepDefault:false, replaceWith:'    '},
5   markupSet:  [
6      {name:'Bold', key:'B', openWith:'(!(<strong>|!|<b>)!)', closeWith:
        '(!(</strong>|!|</b>)!)' },
7       {name:'Italic', key:'I', openWith:'(!(<em>|!|<i>)!)', closeWith:
        '(!(</em>|!|</i>)!)'  },
8       {name:'Stroke through', key:'S', openWith:'<del>', closeWith:
        '</del>' },
9       {separator:'---------------' },
```

```
10        {name:'Bulleted List', openWith:'    <li>', closeWith:'</li>',multi-
          line:true, openBlockWith:'<ul>\n', closeBlockWith:'\n</ul>'},
11        {name:'Numeric List', openWith:'    <li>', closeWith:'</li>', multi-
          line:true, openBlockWith:'<ol>\n', closeBlockWith:'\n</ol>'},
12        {separator:'---------------' },
13        {name:'Picture', key:'P', replaceWith:'<img src="[![Source:!:http:
          //]!]" alt="[![Alternative text]!]" />' },
14        {name:'Link', key:'L', openWith:'<a href="[![Link:!:http:
          //]!]"(!( title="[![Title]!]")!)>', closeWith:'</a>', placeHolder:
          'Your text to link...' },
15        {separator:'---------------' },
16        {name:'Clean', className:'clean', replaceWith:function(markitup)
          { return markitup.selection.replace(/<(.*?)>/g, "") } },
17        {name:'Preview', className:'preview',  call:'preview'}
18    ]
19    }
```

上面这段 JavaScript 代码是编辑器的属性设定内容。第 2 行对 Shift 和回车键同时按下取消默认操作，在编辑器中使用'
\n'来代替。第 3 行对 Ctrl 和回车键同时按下取消默认操作，在编辑器中使用段落标记来代替。第 4 行对 Tab 键的按下取消默认操作，使用 4 个空格代替。第 6 行定义添加粗体字标记按钮。第 7 行定义添加斜体字标记按钮。第 8 行定义添加删除线标记按钮。

第 9 行定义分隔线样式。第 10 行定义添加无序列表样式标记按钮。第 11 行定义添加有序列表样式标记按钮。第 12 行定义分隔线样式。第 13 行定义添加图片元素标记按钮。第 14 行定义能够添加超链接元素标记按钮。第 15 行定义分隔线样式。第 16 行定义清空前后标记按钮功能。第 17 行定义展现预览视图。效果如图 11.1 和图 11.2 所示。

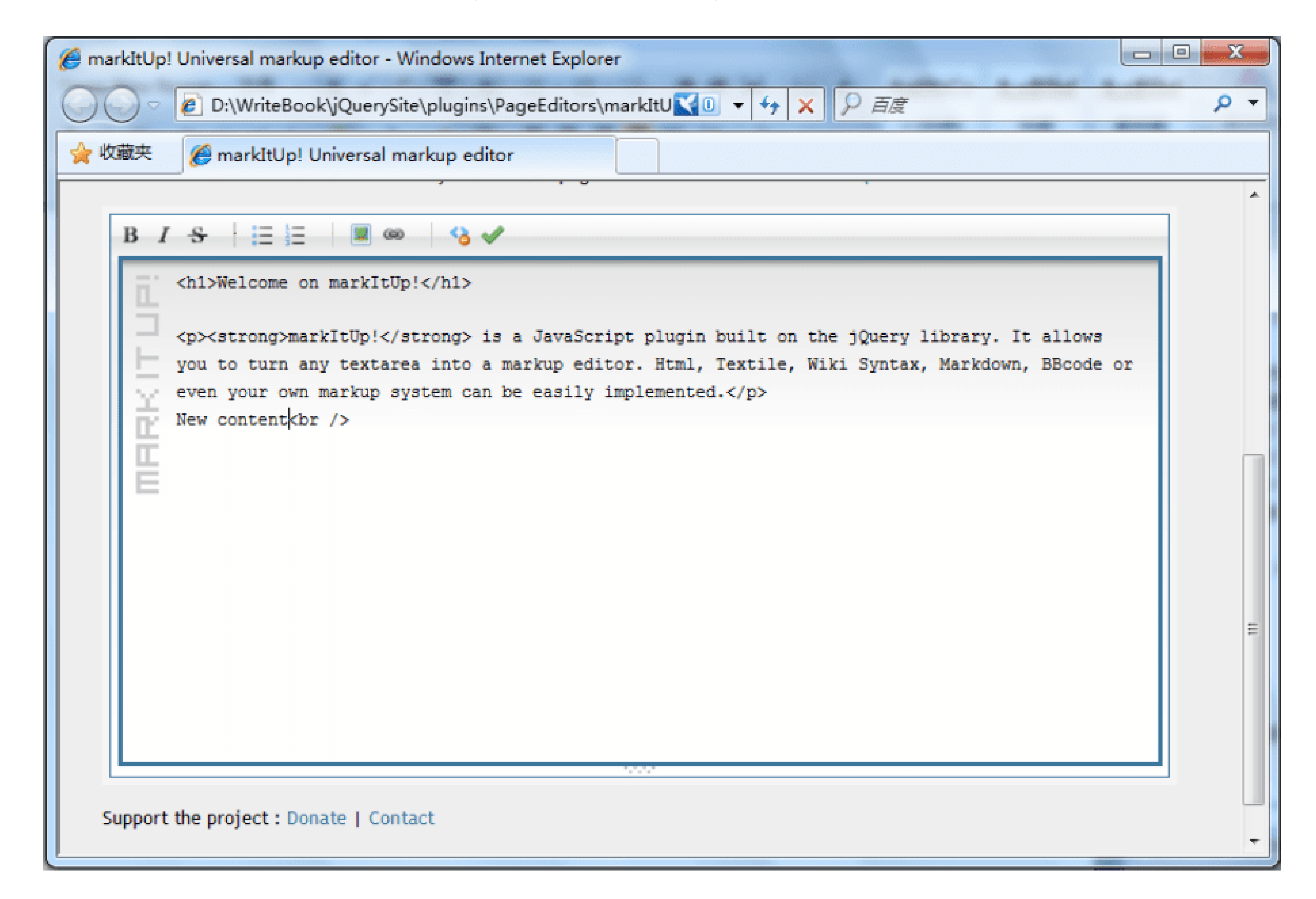


图 11.1　markItUp 编辑器样式

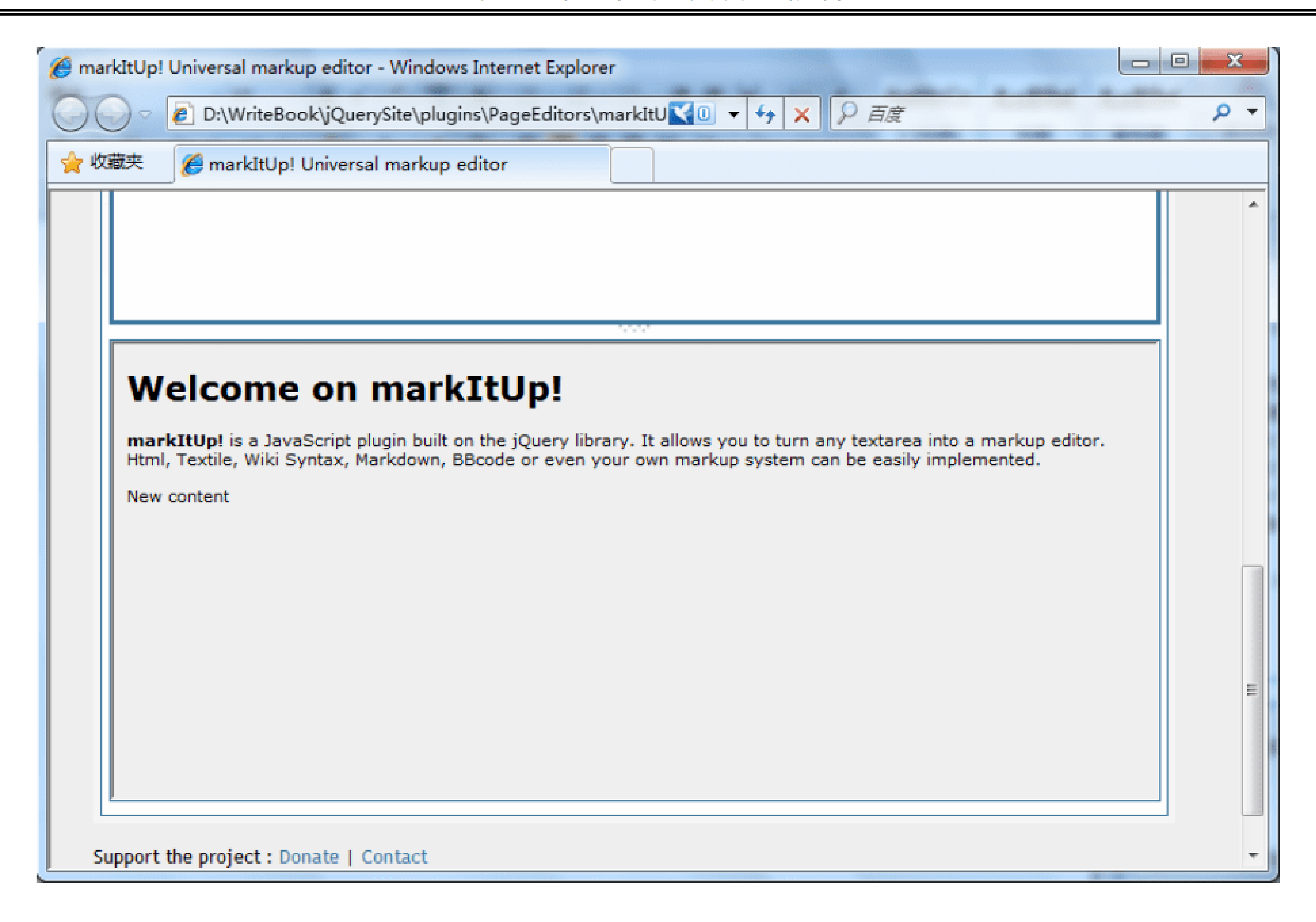


图 11.2　markItUp 预览样式

11.2　jwysiwyg 插件

jwysiwyg 插件是一个所见即所得的 jQuery 页面编辑器。使用这个编辑器需要引用 jQuery 1.3.2 以后版本，还需要 jQuery.UI.Dialog 和 jQuery.UI.Resizable 库文件。这个插件的英文网址为 http://github.com/akzhan/jwysiwyg。

11.2.1　jwysiwyg 插件基本介绍

这个插件主要通过配置属性与函数来控制它的工作，具体属性说明如表 11.3 所示。

表 11.3　jwysiwyg插件的属性说明

属　性	类　型	默 认 值	说　明
html	字符串		在编辑器内的 HTML 源代码
debug	布尔		是否可以调试
css	字符串		编辑器使用的 CSS 样式文件的路径
autoGrow	布尔		框架尺寸自动增长
autoSave	布尔		是否自动保存
brIE	布尔		如果为真，在 IE 中插入 用于创建新行
formHeight	整型		对话框的高
formWidth	整型		对话框的宽
iFrameClass	字符串		使用的 iFrame 的样式类

续表

属　性	类　型	默 认 值	说　明
initialContent	字符串	Initial Content	编辑器的初始内容
maxHeight	整型		自动增长的最大高度
maxLength	整型		编辑器可接收的最多字符个数，但不包括 HTML 标记
messages	jQuery 对象，形式为键/值对		自定义的消息内容
plugins			插件使用基本设置
autoload	布尔或者对象		是否自动加载默认插件对象，或者指定其他加载对象
i18n	布尔或者对象		是否适用国际化，也可指定语言对象
rmMSWordMarkup	布尔		是否移除 MS Word 标记
toolbarHtml	字符串		HTML 源代码
resizeOptions	布尔		是否允许编辑器调整大小
removeHeadings	布尔		是否移除标题
rmUnusedControls	布尔		是否移除未在控制属性集合中未启用的功能（参见表 11.5）
rmUnwantedBr	布尔		是否添加 标记
tableFiller	字符串		添加表格时单元格中的默认文本
events	事件对象		为插件指定事件，键为事件名，值为回调函数
controls	对象		功能按钮集合

插件具有内置函数，为我们操作插件提供了方便，如表 11.4 所示。

表 11.4　jwysiwyg插件的内置函数说明

函　数	说　明
addControl(name,settings)	添加控制按钮
clear	清除内容
createLink(szURL)	创建超链接
destroy	销毁编辑器对象
document	编辑器中的文档
getContent	获取编辑器内容
insertImage(szURL,attribute)	插入图片
insertTable(colCount,rowCount,filler)	添加表格
removeFormat	移除格式
save	存储编辑器中文本的改变
selectAll	选中编辑器中所有内容
setContent	设置编辑器中文本内容

编辑器中包含一些控制器按钮，为用户提供文本编辑操作，效果图 11.3 是常用按钮样式。按钮说明如表 11.5 所示。



图 11.3　常用功能按钮

表 11.5　jwysiwyg插件的按钮说明

按　钮	说　明	按　钮	说　明
bold	粗体字	insertHorizontalRule	插入水平线
italic	斜体字	createLink	创建超链接
strikeThrougn	删除线	insertImage	插入图片
underline	下划线	h1	添加 h1 标题
justifyLeft	左对齐	h2	添加 h2 标题
justifyCenter	居中对齐	h3	添加 h3 标题
justifyRight	右对齐	paragraph	添加段落或者 h1～h6 标题
justifyFull	全对齐	cut	剪切
indent	文本缩进	copy	复制
outdent	文本突出	paste	粘贴
subscript	文本下标	increaseFontSize	字号变大
supscript	文本上标	decreaseFontSize	字号变小
undo	撤销操作	html	文本区域的 HTML 源代码
redo	重做	removeFormat	移除所有格式
insertOrderedList	插入有序列表	insertTable	插入表格
insertUnorderedList	插入无序列表		

下面通过示例，学习如何使用该插件。本书所有的效果图都是基于 IE8 的，这个插件如果在 IE 环境下使用，需要打开 jquery.wysiwyg.css 文件，将 jquery.wysiwyg.gif 修改为 jquery.wysiwyg.no-alpha.gif，才能看到正常效果。

11.2.2　基本应用

这个示例使用了不带任何参数设置的初始化参数。在这里使用默认编辑器的形态，加载了常用按钮，使用基本功能。在 HTML 文件头部分需要引入下列文件。

```
<link rel="stylesheet" type="text/css" href="../lib/blueprint/screen.css"
media="screen, projection" />
<link rel="stylesheet" type="text/css" href="../lib/blueprint/print.css"
media="print" />
<!--[if lt IE 8]><link rel="stylesheet" href="../lib/blueprint/ie.css"
type="text/css" media="screen, projection" /><![endif]-->
<link rel="stylesheet" href="../../jquery.wysiwyg.css" type="text/css"/>
                                          //插件的 CSS 文件
<script type="text/JavaScript" src="../lib/jquery.js"></script>
<script  type="text/JavaScript"  src="../../jquery.wysiwyg.js"></script>
                                          //插件文件
<script type="text/JavaScript" src="../../controls/wysiwyg.image.js"></script>
```

```
                                                            //插件图片操作文件
<script type="text/JavaScript" src="../../controls/wysiwyg.link.js"></script>
                                                            //插件链接操作文件
<script type="text/JavaScript" src="../../controls/wysiwyg.table.js"></script>
                                                            //插件表格文件
```

JavaScript 功能代码相对简单：

```
1    <script type="text/JavaScript">
2    (function($) {
3        $(document).ready(function() {
4            $('#wysiwyg').wysiwyg();    //插件初始化
5        });
6    })(jQuery);
7    </script>
```

第 4 行调用插件初始化函数。效果如图 11.4 所示。

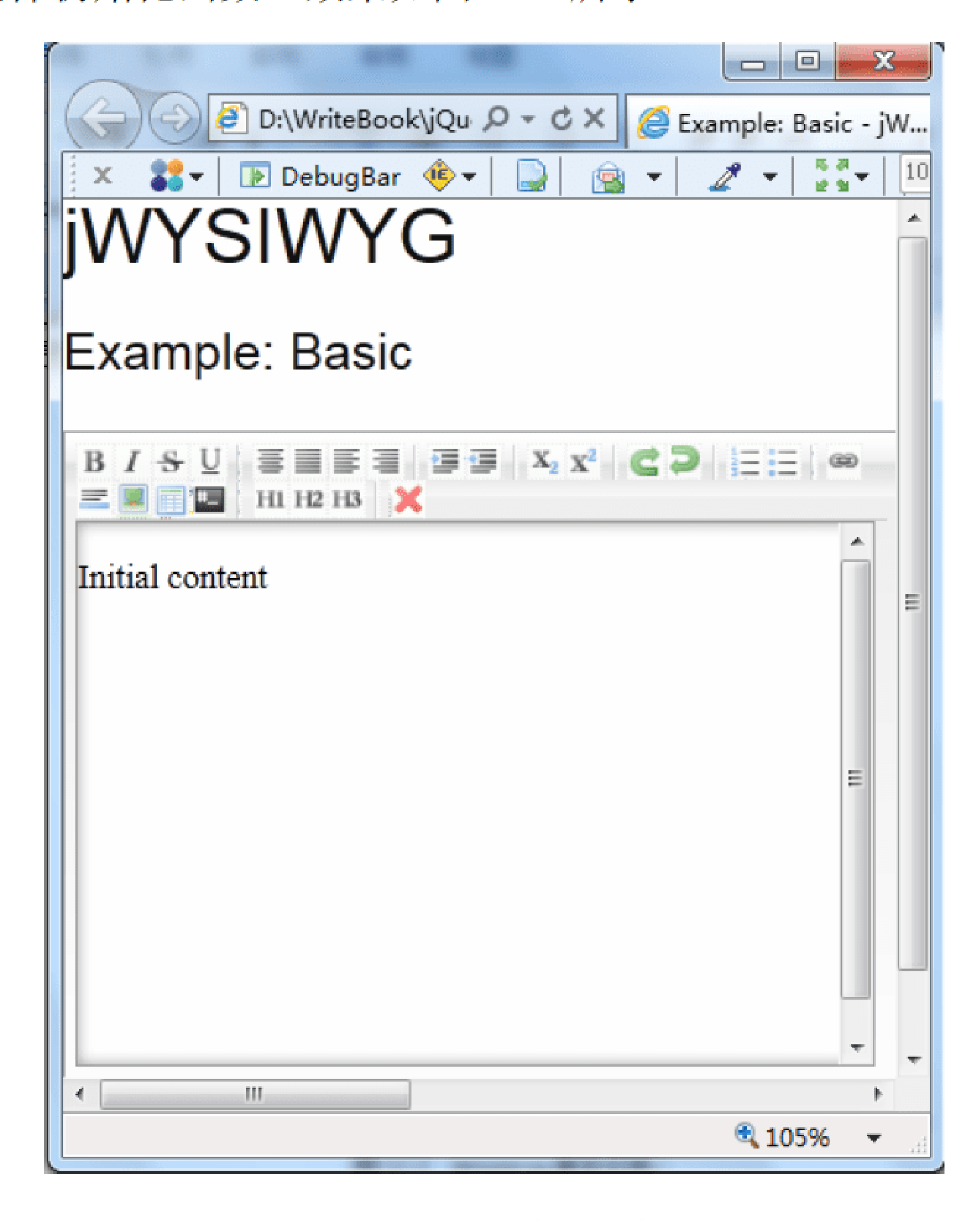


图 11.4 jwysiwyg 插件的基本应用效果

11.2.3 全功能编辑器

这个示例在插件初始化过程中将插件所支持的功能全部加载。这里使用了前面介绍的关于插件中添加标记按钮的属性，并为一些标记属性（如标题按钮）指定了依据不同浏览器实现功能的样式，以及插件的单击实现效果。JavaScript 功能代码如下：

```
1 <script type="text/JavaScript">
2 (function($) {
3   $(document).ready(function() {
4       $('#wysiwyg').wysiwyg({            //配置插件所有功能
5         controls: {
6           bold          : { visible : true },
7           italic        : { visible : true },
8           underline     : { visible : true },
9           strikeThrough : { visible : true },
10
11          justifyLeft   : { visible : true },
12          justifyCenter : { visible : true },
13          justifyRight  : { visible : true },
14          justifyFull   : { visible : true },
15          indent  : { visible : true },
16          outdent : { visible : true },
17          subscript   : { visible : true },
18          superscript : { visible : true },
19
20          undo : { visible : true },
21          redo : { visible : true },
22
23          insertOrderedList    : { visible : true },
24          insertUnorderedList  : { visible : true },
25          insertHorizontalRule : { visible : true },
26          h4: {
27              visible: true,
28              className: 'h4',
29              command: ($.browser.msie || $.browser.safari) ?
                    'formatBlock' : 'heading',
30              arguments: ($.browser.msie || $.browser.safari) ? '<h4>' : 'h4',
31              tags: ['h4'],
32              tooltip: 'Header 4'
33          },
34          h5: {
35              visible: true,
36              className: 'h5',
37              command: ($.browser.msie || $.browser.safari) ?
                'formatBlock' : 'heading',
38              arguments: ($.browser.msie || $.browser.safari) ? '<h5>' : 'h5',
39              tags: ['h5'],
40              tooltip: 'Header 5'
41          },
42          h6: {
43              visible: true,
```

```
44              className: 'h6',
45              command: ($.browser.msie || $.browser.safari) ?
                'formatBlock' : 'heading',
46              arguments: ($.browser.msie || $.browser.safari) ? '<h6>' : 'h6',
47              tags: ['h6'],
48              tooltip: 'Header 6'
49          },
50
51          cut   : { visible : true },
52          copy  : { visible : true },
53          paste : { visible : true },
54          html  : { visible: true },
55          increaseFontSize : { visible : true },
56          decreaseFontSize : { visible : true },
57          exam_html: {
58              exec: function() {
59                  this.insertHtml('<abbr title="exam">Jam</abbr>');
60                  return true;
61              },
62              visible: true
63          }
64        },
65        events: {
66          click: function(event) {    //插件单击事件
67              if ($("#click-inform:checked").length > 0) {
68                  event.preventDefault();
69                  alert("You have clicked jWysiwyg content!");
70              }
71          }
72        }
73      });
74      $('#wysiwyg').wysiwyg("insertHtml", "Sample code");
                                              //设定插件内文本内容
75  });
76  })(jQuery);
77  </script>
```

上述代码第 5 行开始设定编辑器的控制按钮功能。第 6～25 行设定粗体、斜体、下划线、删除线、左对齐、右对齐、居中对齐、全对齐、内缩进、外突出、上标、下标、取消、重做、有序列表插入、无序列表插入、水平线等按钮全部显示。

第 26～33 行设定<h4>标题按钮相关属性，按钮可见，样式类名为 h4，如果当前是 IE 浏览器则命令名称为格式化块，如果当前浏览器是 Safari 浏览器则为标题，如果当前浏览器是 IE 浏览器则参数内容为<h4>，如果当前浏览器是 Safari 浏览器则为 h4，标记['h4']，工具提示'Header 4'。第 34～41 行设定<h5>标题按钮相关属性。

第 42～49 行设定<h6>标题按钮相关属性，相关内容和<h4>的设定类似。第 51～56 行设定剪切、复制、粘贴、HTML 源代码、字体放大、字体缩小按钮全部显示。第 57～63 行设定 exam_html 按钮的功能，向文本部分添加 HTML 代码<abbr title="exam">Jam</abbr>，按钮可见。第 65～72 行添加编辑器文本部分单击事件，如果选中复选框则单击文本的时候跳出信息提示对话框。

效果如图 11.5 和图 11.6 所示。

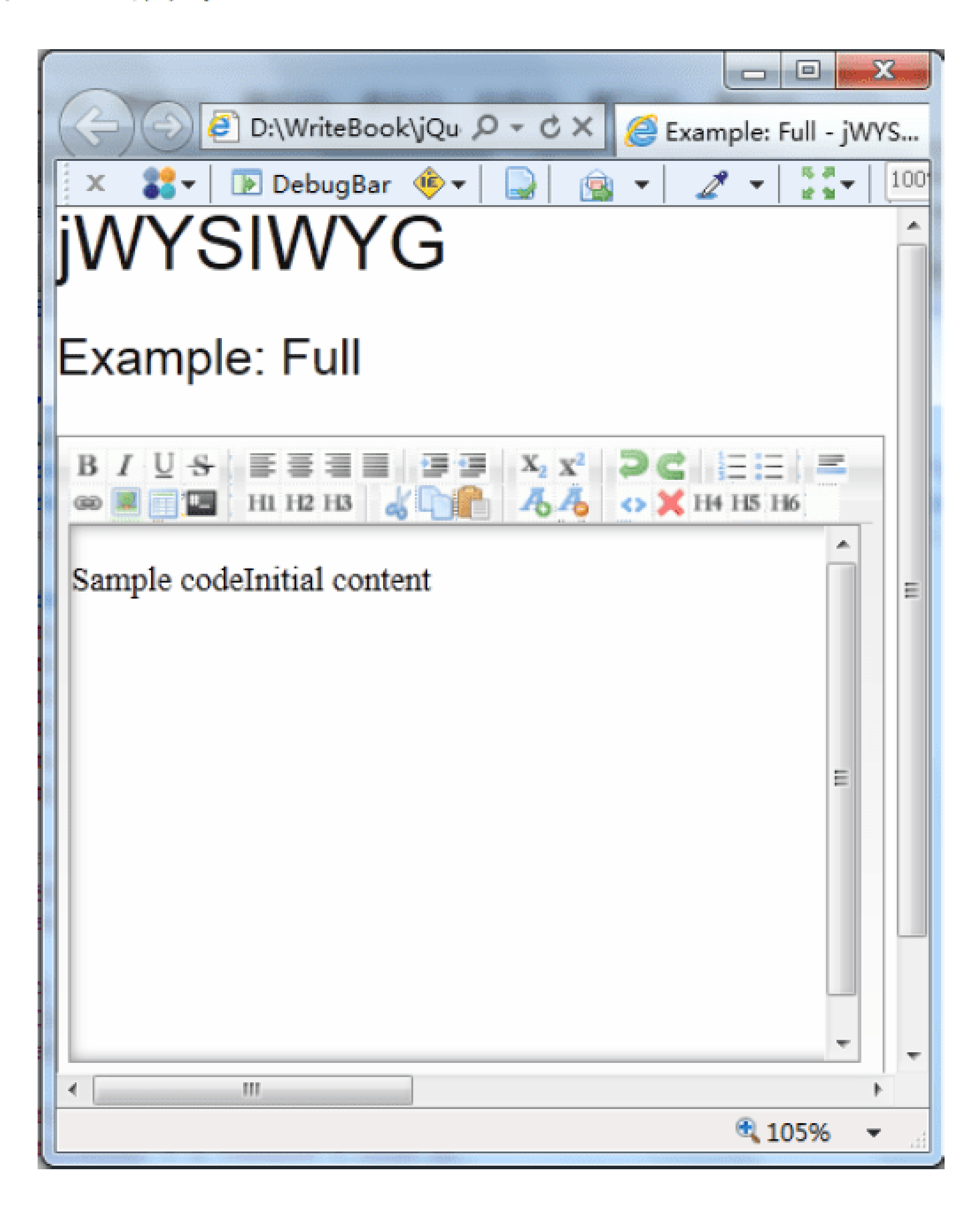


图 11.5　全功能编辑器

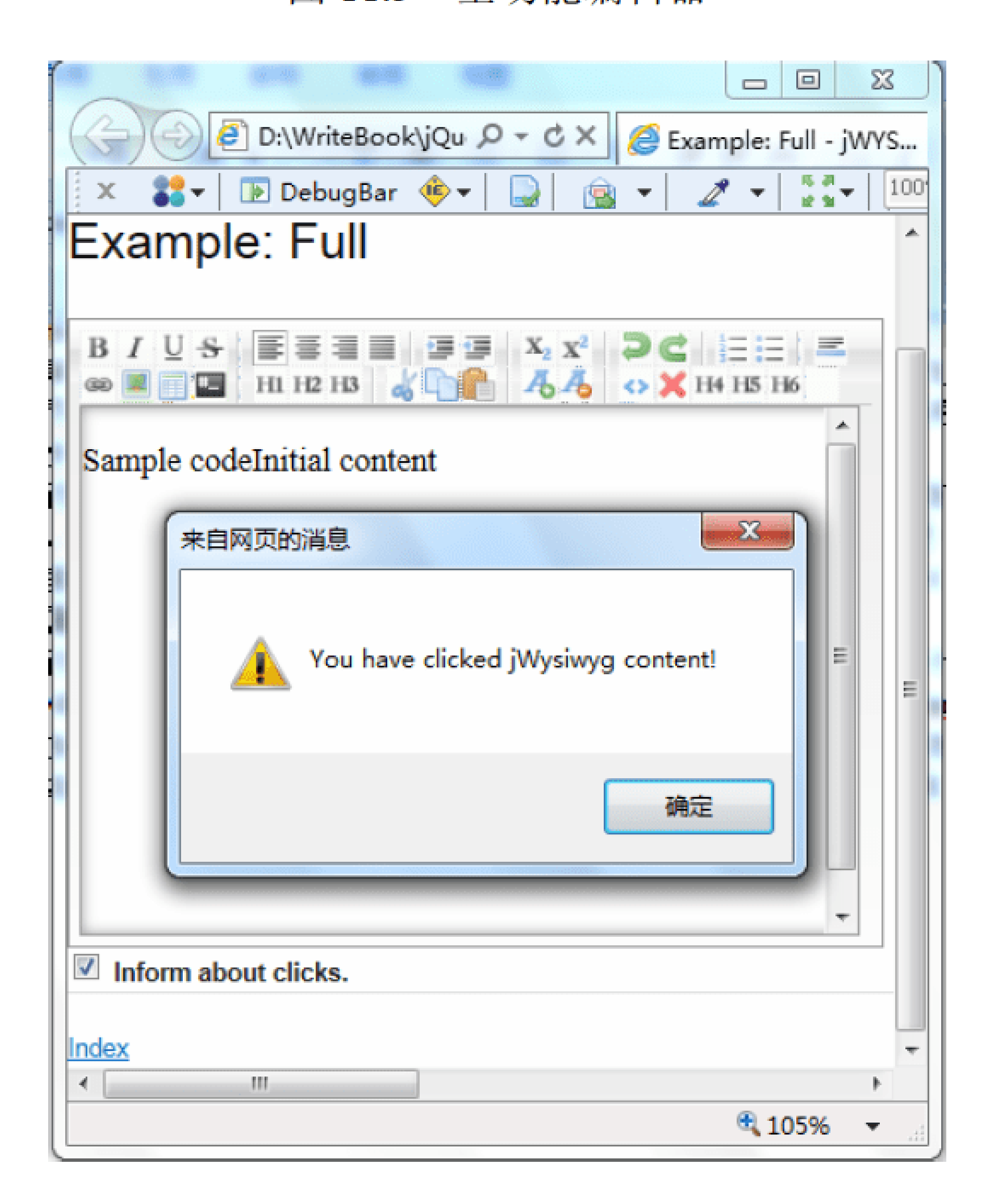


图 11.6　单击文本跳出提示对话框

11.2.4　可调整大小的编辑器

这个示例利用了 jQuery UI 库的调整元素大小的库文件，并对 jwysiwyg 插件的相关属性进行了调整，设定了插件的可调整大小属性，但是要在 HTML 文件头部多引入 jQuery UI 库文件：

```
<script type="text/JavaScript" src="../lib/ui/jquery.ui.core.js"></script>
<script type="text/JavaScript" src="../lib/ui/jquery.ui.widget.js"></script>
<script type="text/JavaScript" src="../lib/ui/jquery.ui.mouse.js"></script>
<script type="text/JavaScript" src="../lib/ui/jquery.ui.resizable.js"></script>
```

JavaScript 代码如下：

```
1  <script type="text/JavaScript">
2   (function ($) {
3       $(document).ready(function () {
4           $('#wysiwyg').wysiwyg({
5               resizeOptions: {},      //调整插件大小选项
6               controls: {
7                   html: { visible: true }
8               }
9           });
10      });
11  })(jQuery);
12  </script>
```

上述代码第 5 行设定编辑器可调整大小，在编辑器的右下角能看到调整大小的箭头。效果如图 11.7 和图 11.8 所示。

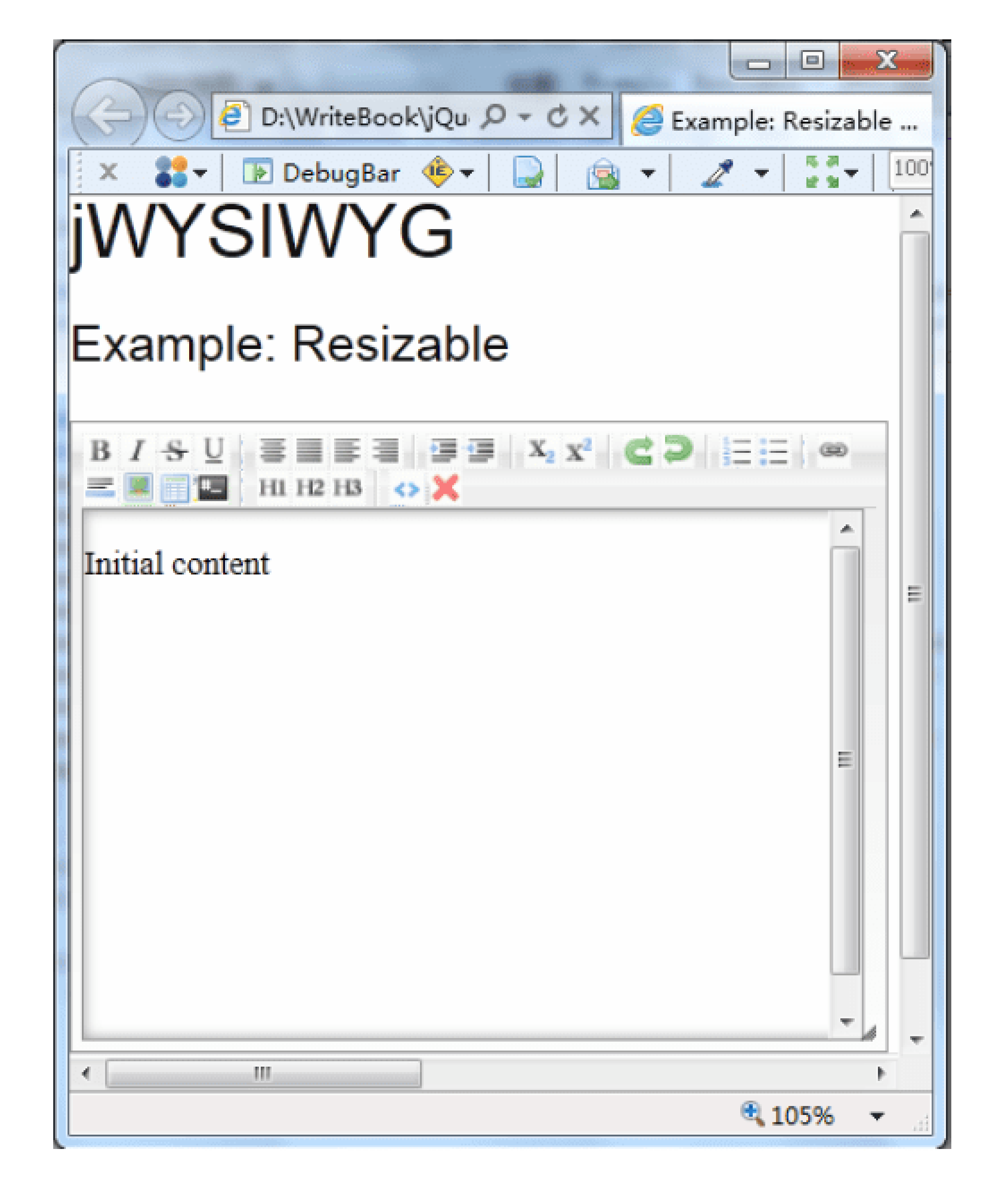


图 11.7　编辑器原始尺寸

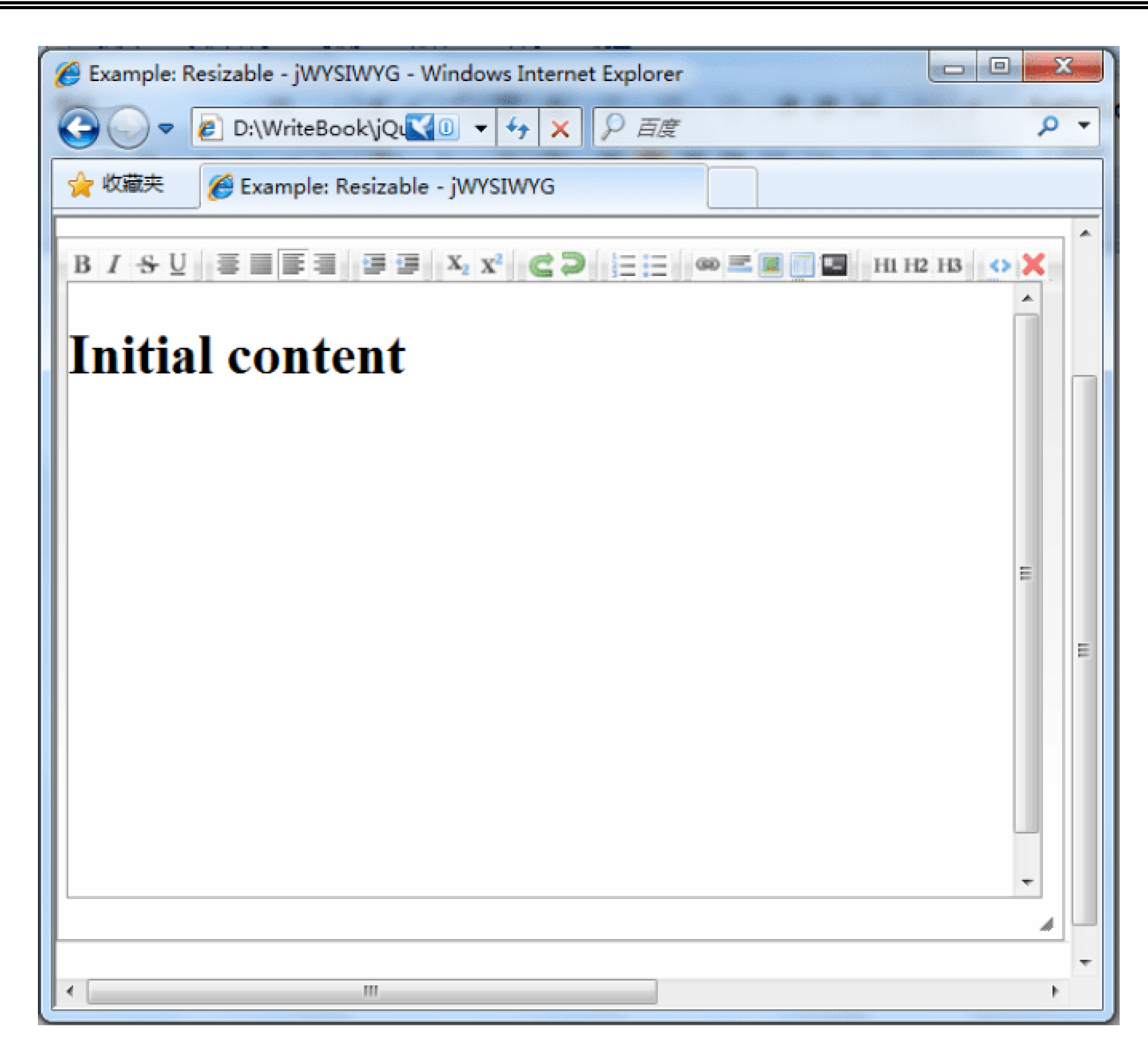


图 11.8　调整大小后效果

11.2.5　编辑器内容的左右移动

这个示例在编辑器的控制按钮集合中加入了左右移动按钮，可以使编辑器内容从左边框移至右边框，也可反方向移动。实际上这里使用了插件中内容的左右对齐属性。JavaScript 代码如下：

```
1  <script type="text/JavaScript">
2   (function ($) {
3       $(document).ready(function () {
4           $('#wysiwyg').wysiwyg({
5               controls: {
6                   rtl: { visible: true },     //左对齐功能可见
7                   ltr: { visible: true }      //右对齐功能可见
8               }
9           });
10      });
11  })(jQuery);
12 </script>
```

上述代码第 6、7 两行设定了 Right to Left 和 Left to Right 按钮为显示状态。效果如图 11.9 和图 11.10 所示。

11.2.6　根据内容自动调整大小的编辑器

这个示例使用了编辑器的自动增长属性，使编辑器可以根据内容自动增长。这里使用

了前面介绍的自动增长属性及最大高度属性。JavaScript 代码如下：

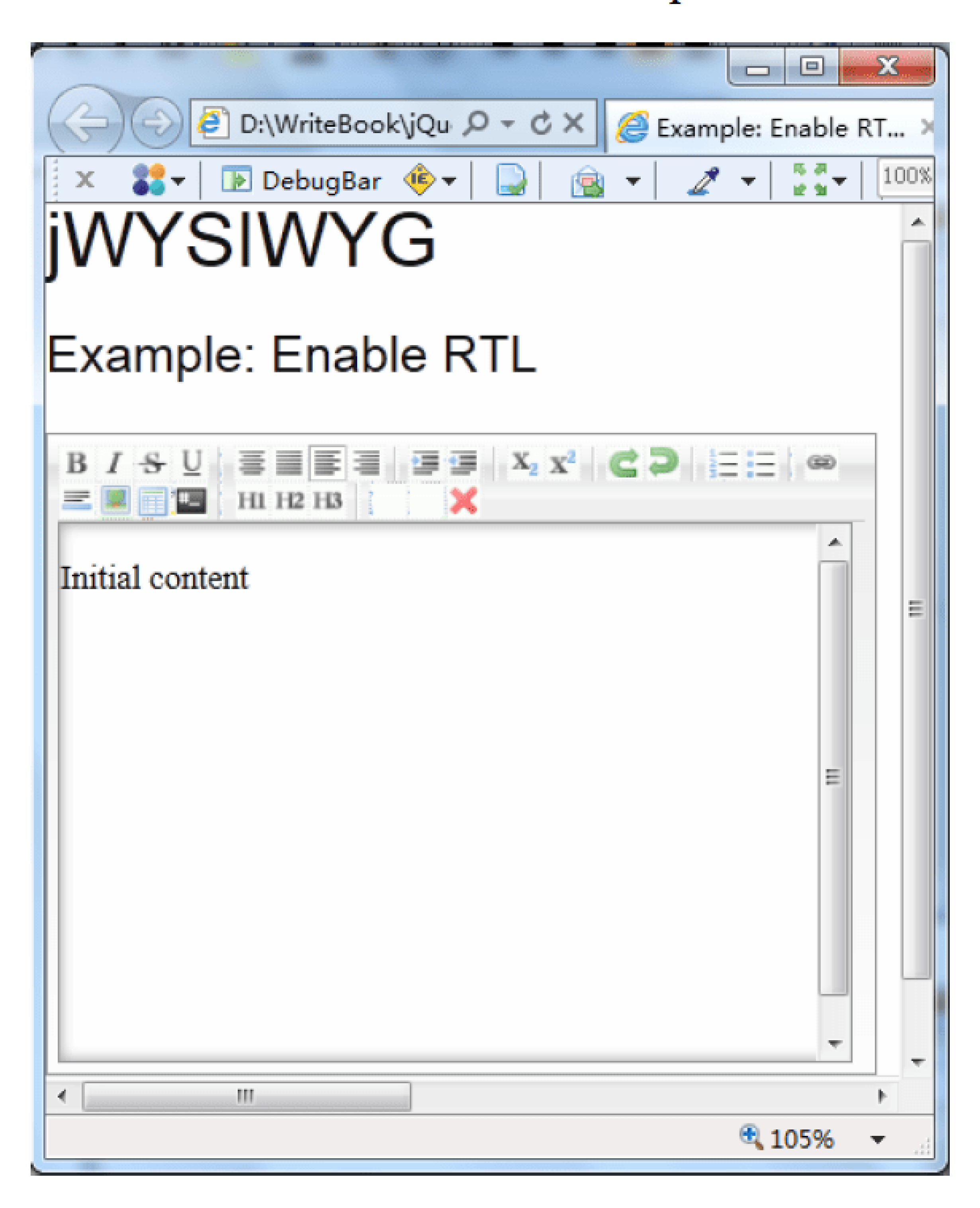


图 11.9　编辑器内容左右移动按钮效果一

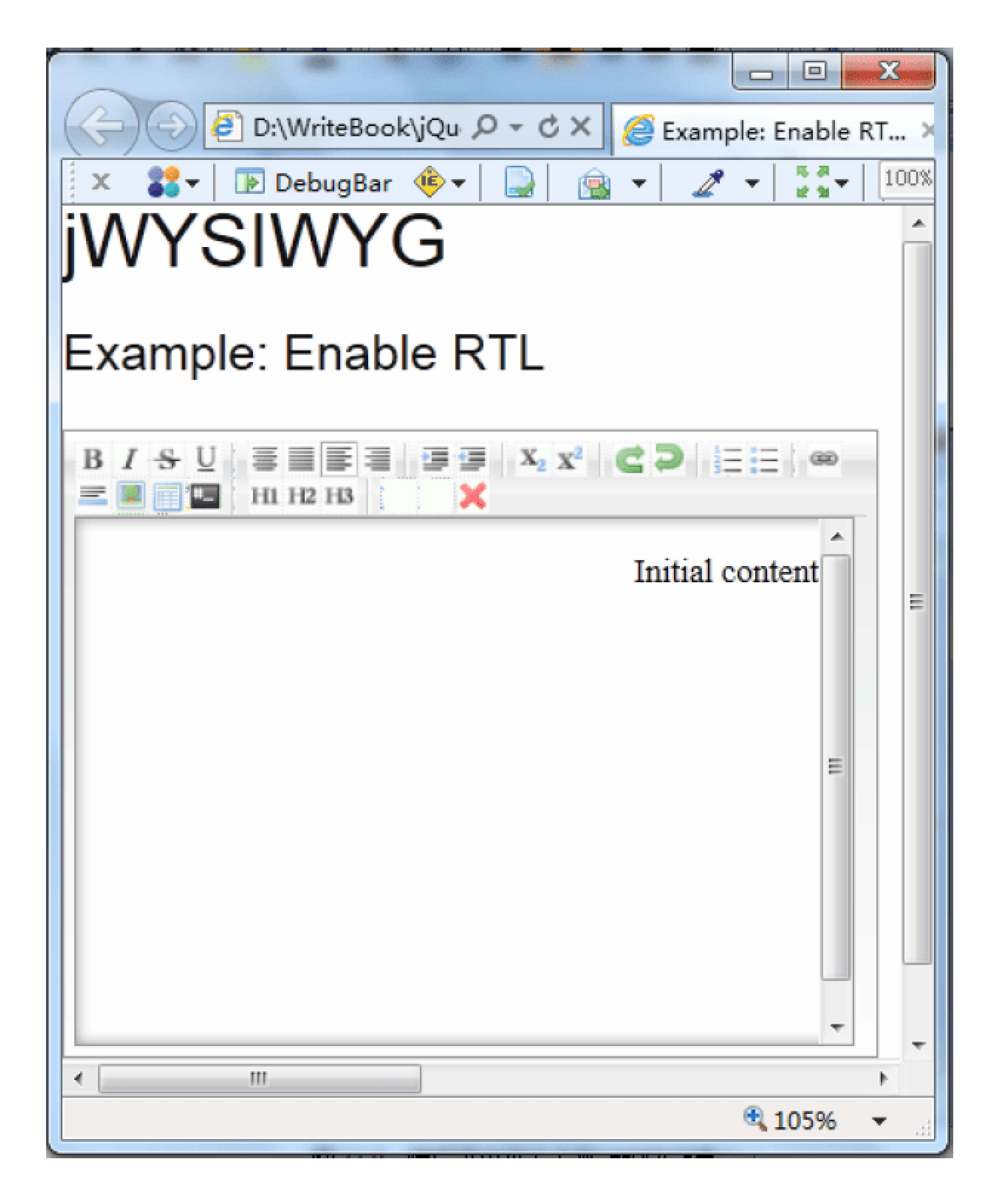


图 11.10　编辑器内容左右移动按钮效果二

```
1 <script type="text/JavaScript">
2   (function ($) {
3       $(document).ready(function () {
4           $('#wysiwyg').wysiwyg({
5               autoGrow: true, //插件可自动增长
6               maxHeight: 600  //插件最大高度为 600 像素
7           });
8       });
9   })(jQuery);
10 </script>
```

上述代码第 5 行设定编辑器可自动增长，第 6 行限制了自动增长的最大高度为 600。效果如图 11.11 所示。

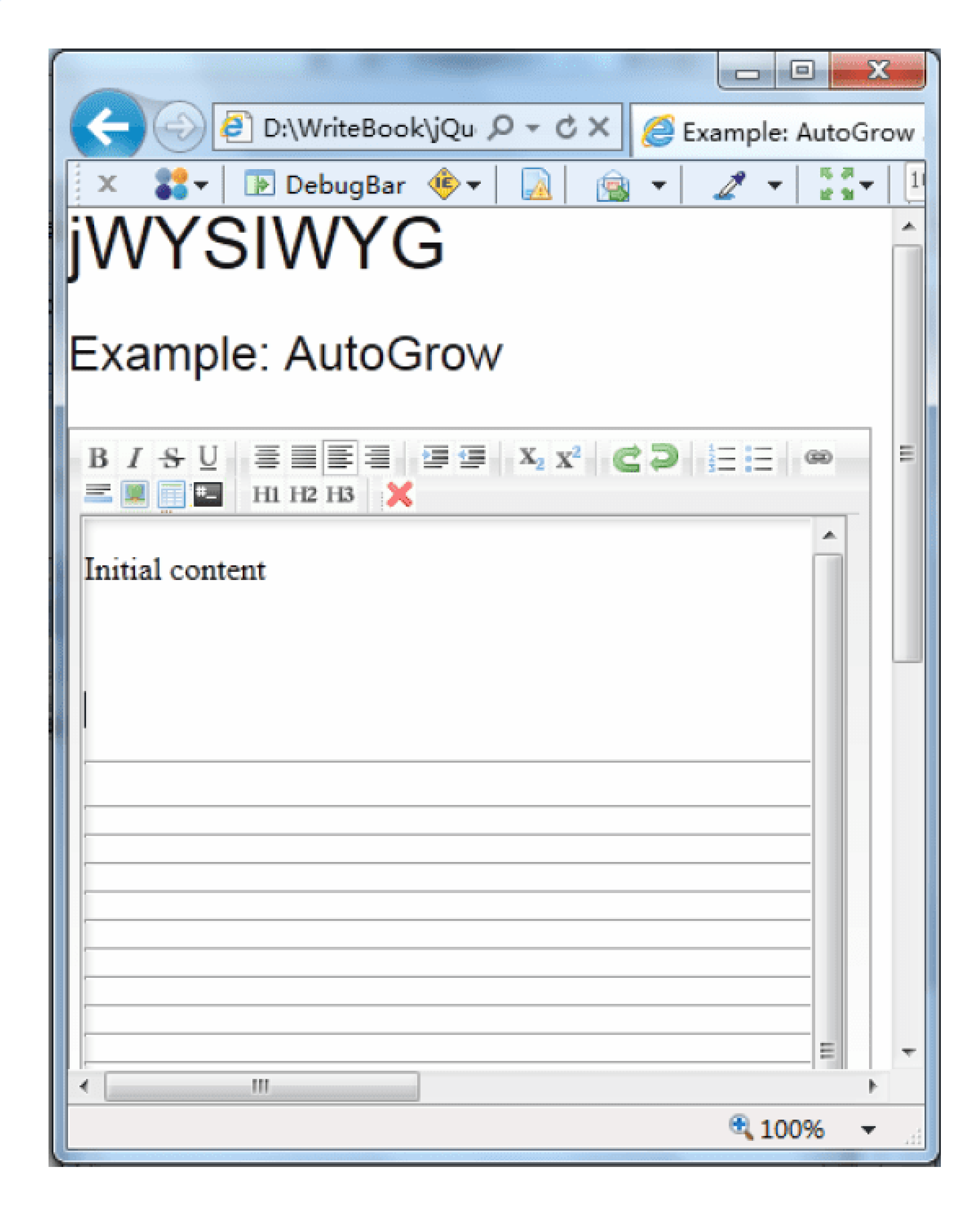


图 11.11　自动增长大小的编辑器

11.2.7　自定义功能控制按钮

这个示例使用了可以自定义功能的控制按钮，这个按钮的功能是在编辑器内容中添加引号。这里添加了一个自定义的内缩进标记按钮，并使用了 blockquote 标记包装我们选择的文本。JavaScript 代码如下：

```
1 <script type="text/JavaScript">
2   (function($) {
3       $(document).ready(function() {
4           $("textarea").wysiwyg({
5               rmUnusedControls: true,                //隐藏不使用的按钮
6               controls: {
```

```
7                   bold: { visible : true },           //粗体功能
8                   html: { visible : true },          //HTML 源代码查看功能
9                   insertOrderedList: { visible : true }, //插入有序列表功能
10                  removeFormat: { visible : true }      //移除原有格式功能
11              }
12          });
13          $("input[name=add]").click(function () {
14              $("textarea").wysiwyg("addControl", "quote", {
15                  groupIndex: 2,
16                  icon: './../tests/images/quote02.gif',
17                  tooltip: 'Quote',
18                  tags: ['blockquote'],
19                  exec: function () {
20                      var range   = this.getInternalRange(),
21                          common  = range.commonAncestorContainer,
22                          blockquote = this.dom.getElement("blockquote");
                              //如果选中的是文本内容，则使用标记对其进行包装
23                      if (common.nodeType === 3) {
24                          common = common.parentNode;
25                      }
26                      if (blockquote && $(blockquote).hasClass("quote")) {
27                          $(common).unwrap();
28                  }
29                      else {
30                          if ("body" !== common.nodeName.toLowerCase()) {
31                              $(common).wrap("<blockquote class='quote' />");
32                          }
33                      }
34                  },
35                  callback: function (event, Wysiwyg) {
36                      alert("callback triggered!");
37                  }
38              });
39          });
40          $("input[name=makeBold]").click(function () {
41              $("textarea").wysiwyg("triggerControl", "bold");
42          });
43      });
44  })(jQuery);
45 </script>
```

上述代码第 5 行设定不使用的按钮不显示。第 7～10 行设定使用粗体、查看 HTML 源代码、插入有序列表、移除格式等功能按钮。第 13 行设定当点击添加新控制的时候会在编辑器的工具栏上添加一个实现<blockquote>标记的按钮，选中内容并单击这个按钮，会产生右缩进效果。效果如图 11.12 和图 11.13 所示。

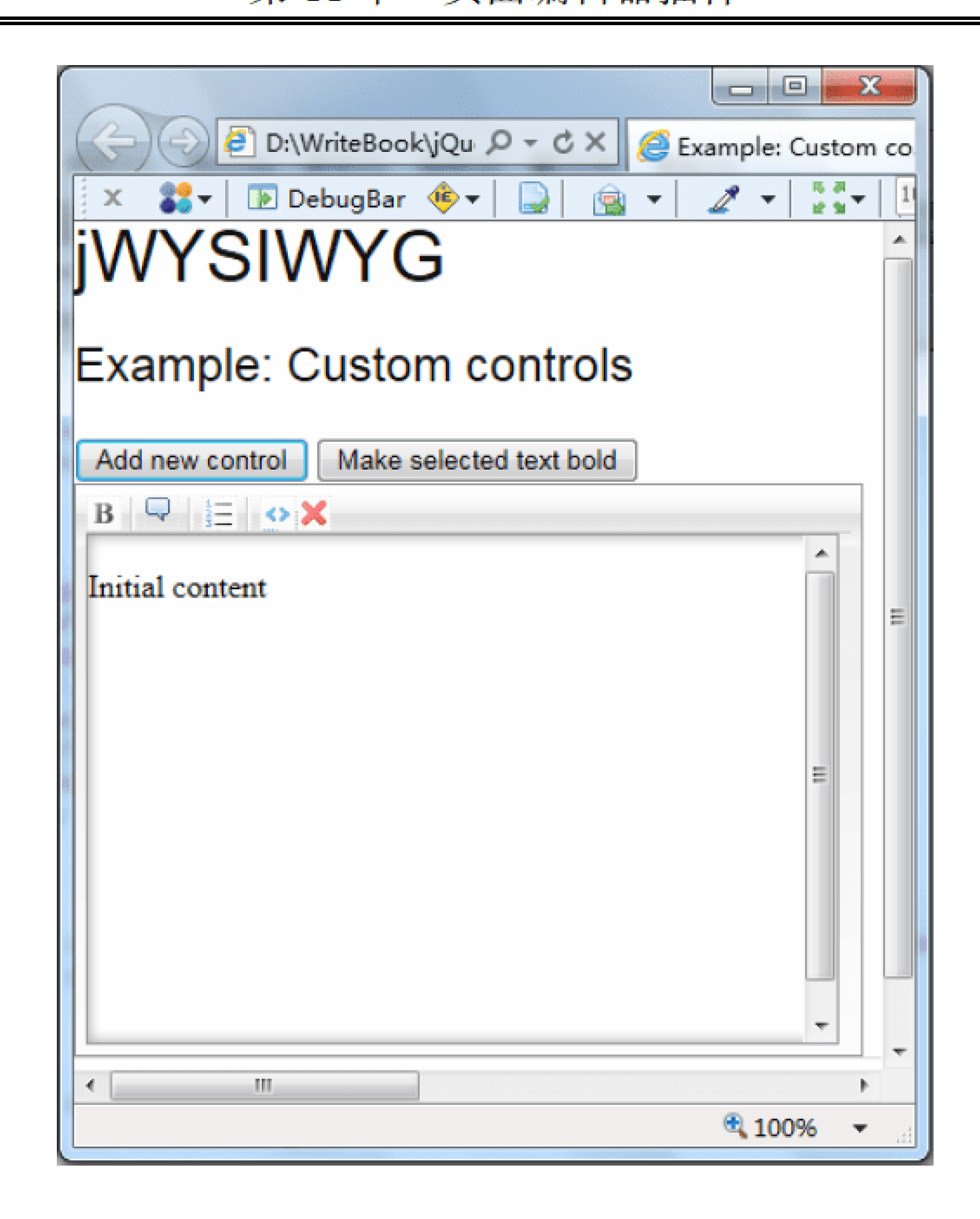


图 11.12　添加右缩进功能按钮

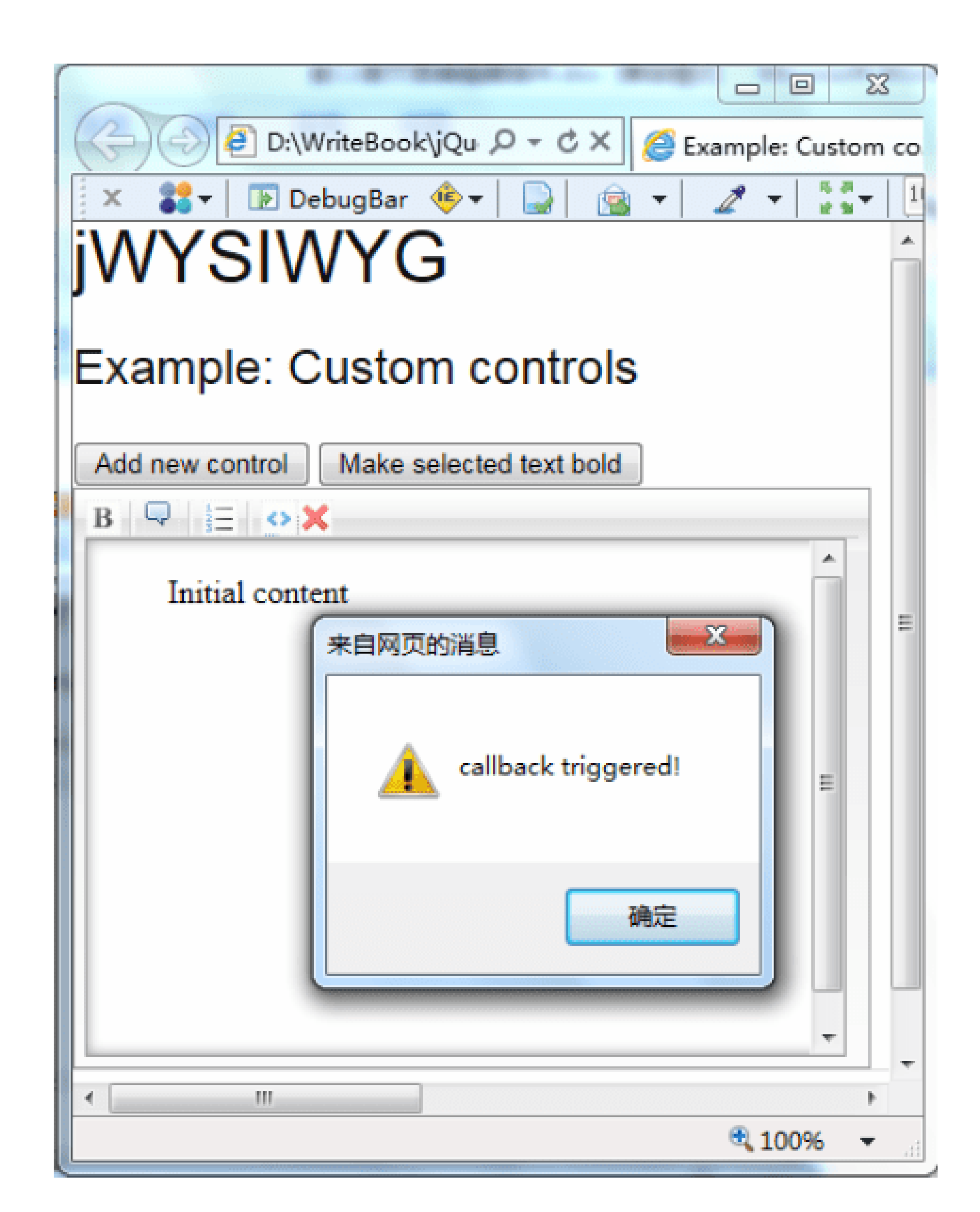


图 11.13　使用右缩进功能按钮

11.3　HtmlBox 插件

HtmlBox 插件是一个跨浏览器、交互式、开源的、所见即所得的 jQuery 编辑器。这个插件安装简单直观，配置方便，适用于 CMS、论坛、用户留言、联系信息等应用。它的英文网址为 http://remiya.com/htmlbox/。

HtmlBox 插件的特点如下。

（1）易整合。

（2）多浏览器支持。

（3）占用空间小。

（4）XHTML 输出。

（5）支持 Ajax。

（6）更换外观方便。

（7）自定义图标。

下面介绍这个插件的基本使用方法。

11.3.1　如何安装

插件的安装实际就是插件文件的引入。在引入这个插件之前同样需要先引入 jQuery 库文件。这个插件使用的 jQuery 库文件版本较低。

（1）引入 JS 文件：

```
<script language="JavaScript" src="jquery-1.3.2.min.js" type="text/
JavaScript"></script>
<script language="JavaScript" src="htmlbox.colors.js" type="text/
JavaScript"></script>                 //插件颜色操作文件
<script language="JavaScript" src="htmlbox.styles.js" type="text/
JavaScript"></script>                 //插件样式操作文件
<script language="JavaScript" src="htmlbox.syntax.js" type="text/
JavaScript"></script>                 //插件语法检查文件
<script language="JavaScript" src="xhtml.js" type="text/JavaScript">
</script>                             //XHtml 标准文件
<script language="JavaScript" src="htmlbox.min.js" type="text/
JavaScript"></script>                 //插件核心文件
```

（2）创建一个文本区域：

```
<textarea id='ha'></textarea>
```

（3）设置运行插件的属性特点。属性的使用方法参考下面介绍的示例。

11.3.2　属性特点

在运行插件时需要对插件进行初始化，即配置运行属性，如表 11.6 所示。

表 11.6　HtmlBox属性说明

属　　性	类　　型	默　认　值	说　　明
idir	字符串	./image/	需要使用图片的路径位置，可以是绝对路径，也可以是相对 URL 地址
about	布尔	true	是否使用 about 说明对话框

续表

属　　性	类　　型	默　认　值	说　　明
icons	字符串	default	设置需要使用的图标
skin	字符串	default	设置皮肤样式
toolbars	数组	["bold","italic","underline"]	设置工具栏内容
output	字符串	xhtml	设置是哪种输出内容标准
css	字符串	body{margin:3px;font-family:vendana;font-size:11;}p{margin:0px;}	设置使用的 CSS 样式
toolbar_height	整型	16	工具栏的高
tool_height	整型	16	工具的高
tool_width	整型	16	工具的宽
tool_image_height	整型	16	工具栏上图片的高
tool_image_width	整型	16	工具栏上图片的宽
success	函数	function(data){alert(data);}	一个成功的 Ajax 调用返回后调用的函数
error	函数	function(a,b,c){return this;}	当 Ajax 调用出错时调用的函数

11.3.3　内置函数

HtmlBox 通过内置函数实现特定功能，如表 11.7 所示。

表 11.7　HtmlBox内置函数

函　　数	说　　明
get_html()	返回编辑器中内容的 HTML 源代码，如果 xhtml.js 被引入，则返回 XHTML 源代码
get_text()	返回在 HTML 标记中的文本内容
set_text()	设置编辑器中的文本
get(url,data)	使用 get 方式利用 Ajax 向服务器提交编辑器中的数据，如果 data 为空，则使用 get_text()获取数据
post(url,data)	使用 post 方式利用 Ajax 向服务器提交编辑器中的数据，如果 data 为空，则使用 get_text()获取数据
success(fn)	如果上两个方法成功返回，则使用函数处理，默认警告浏览返回
error(fn)	如果 Ajax 返回失败，则使用函数处理
cmd(cmd,arg)	执行一个指定的命令
change(fn)	文本改变时使用对应函数处理
remove()	移除编辑器实例
wap_tags(start,end)	在选中的文本外包裹标记

11.3.4　编辑器可使用的工具

HtmlBox 可支持的对文本进行加工的工具如表 11.8 所示。

表 11.8　HtmlBox工具说明

工　　具	说　　明	工　　具	说　　明
cut	从编辑器中剪切选中文本到剪切板中	justify	选中文本分散对齐
copy	从编辑器中复制选中文本到剪切板中	ol	选中行添加有序列表
paste	粘贴剪切板中的内容到编辑器的光标处	ul	选中行添加无序列表
bold	将选中的文本改成粗体	indent	选中文本右缩进
italic	将选中的文本改成斜体	outdent	选中文本突出
underline	在选中的文本下添加下划线	hyperlink	在选中文本上创建超链接
strike	在选中的文本上添加删除线	image	插入图片
sup	将选中的文本改成上标形式	code	显示 HTML 代码
sub	将选中的文本改成下标形式	fontsize	改变选中文本的字号大小
left	左对齐选中文本	fontfamily	改变选中文本的字体
right	右对齐选中文本	fontcolor	改变选中文本的字体颜色

11.3.5　插件应用举例

（1）全功能编辑器。这个示例使用了编辑器的所有工具功能。这里使用了插件的工具条属性，并在该属性中设定了常用工具按钮，同时设定了插件的外观颜色为蓝色。JavaScript代码如下：

```
1 <script language="JavaScript" type="text/JavaScript">
2 $("#ha").css("height","100%").css("width","100%").htmlbox({
3   toolbars:[
4      [
5       //剪切、复制、粘贴功能
6       "separator","cut","copy","paste",
7       //撤销、重做功能
8       "separator","undo","redo",
9       //粗体字、斜体字、下划线、删除线、上标、下标功能
10      "separator","bold","italic","underline","strike","sup","sub",
11      //左对齐、右对齐、居中对齐、两端对齐功能
12      "separator","justify","left","center","right",
13      //有序列表、无序列表、缩进、突出功能
14      "separator","ol","ul","indent","outdent",
15      //超链接、移除超链接、图片功能
16      "separator","link","unlink","image"
17
18      ],
19      [//显示代码
20      "separator","code",
21       //格式化、字号大小、字体集、字体颜色、高亮功能
22       "separator","formats","fontsize","fontfamily",
23      "separator","fontcolor","highlight",
24      ],
25      [
26      //移除格式、去除标记、分隔线、段落功能
27      "separator","removeformat","striptags","hr","paragraph",
```

```
28       //引号、样式、语法功能
29       "separator","quote","styles","syntax"
30       ]
31   ],
32   skin:"blue"     //皮肤颜色为蓝色
33 });
34 </script>
```

上述代码第 2 行设定编辑器的大小，并调用 HtmlBox 初始化函数。第 3 行使用 toolbars 添加工具集。第 4～31 行将前面介绍的工具添加到工具栏上。其中有些工具在表 11.8 中没有介绍，separator 表示不同工具之间的分隔线，unlink 表示取消超链接，removeformat 表示删除原有格式，striptags 表示去除标记，hr 表示添加分隔线，paragraph 表示添加段落，quote 表示突出选中内容，style 表示 CSS 样式设定，syntax 表示编程语言语法。第 32 行表示插件使用蓝色为主色调。效果如图 11.14 所示。

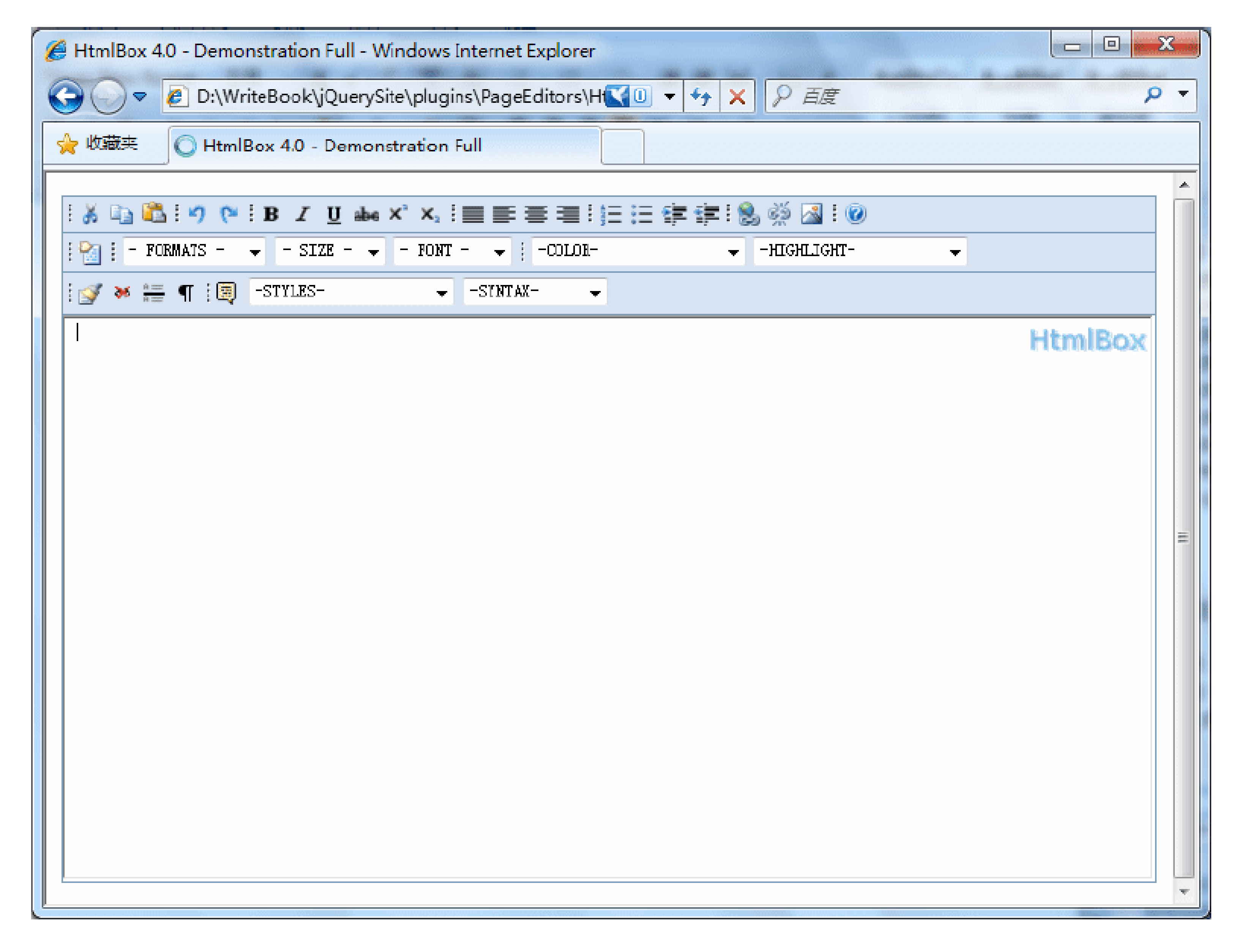


图 11.14　全功能 HtmlBox 编辑器

（2）添加新功能编辑器。这个示例添加了新的图标和图标对应的功能。这个插件同样可以添加自定义的工具按钮并设定它们的功能。新功能的定义在插件初始化的工具栏设定中完成。JavaScript 代码如下：

```
1 <script language="JavaScript" type="text/JavaScript">
2  var hb icon set default = $("#htmlbox icon set default").css("height",
   "100").css("width","600").htmlbox({
3   toolbars:[
4   ["cut","copy","paste","separator dots","bold","italic","underline",
    "strike","separator","sub","sup","separator dots","undo","redo",
    "separator dots","left","center","right","justify","separator dots",
    "ol","ul","indent","outdent","separator dots","link","unlink","image"],
5        ["code","removeformat","striptags","separator dots","quote",
         "paragraph","hr","separator dots",
6            {icon:"new.gif",tooltip:"New",command:function(){hb_icon_
```

```
              set_default.set_text("<p></p>");}},    //新增内容功能
7             {icon:"open.gif",tooltip:"Open",command:function(){alert
              ('Open')}},                            //打开操作功能
8             {icon:"save.gif",tooltip:"Save",command:function(){alert
              ('Save')}}                             //保存操作功能
9         ]
10   ],
11   icons:"default",
12   skin:"default"
13 });
14 </script>
```

上述代码的第 2 行和上面的例子类似，向工具栏中添加工具按钮。第 6 行是新添加一段。第 7 行添加模拟打开功能。第 8 行添加模拟保存功能。效果如图 11.15～图 11.17 所示。

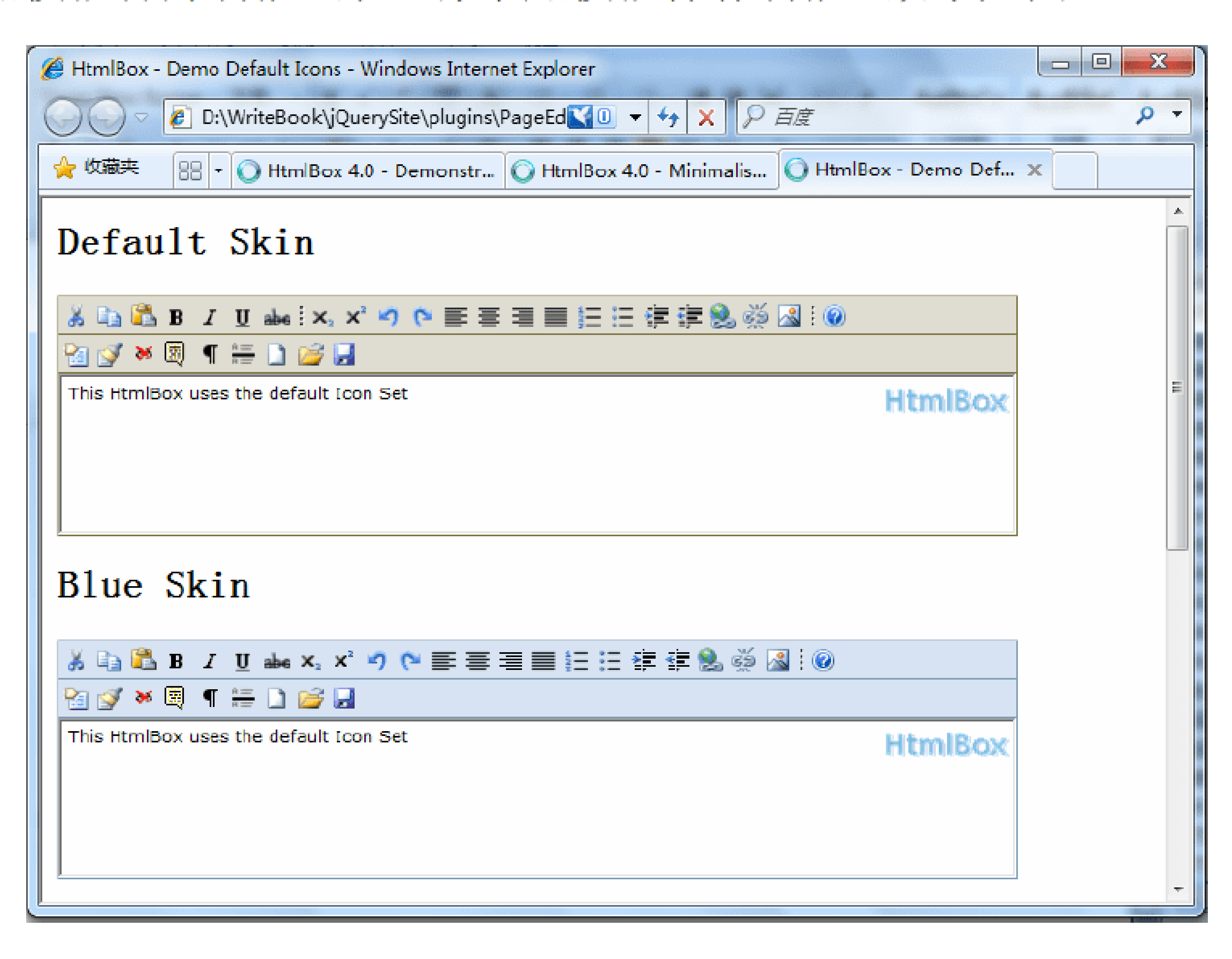


图 11.15　添加新功能的编辑器

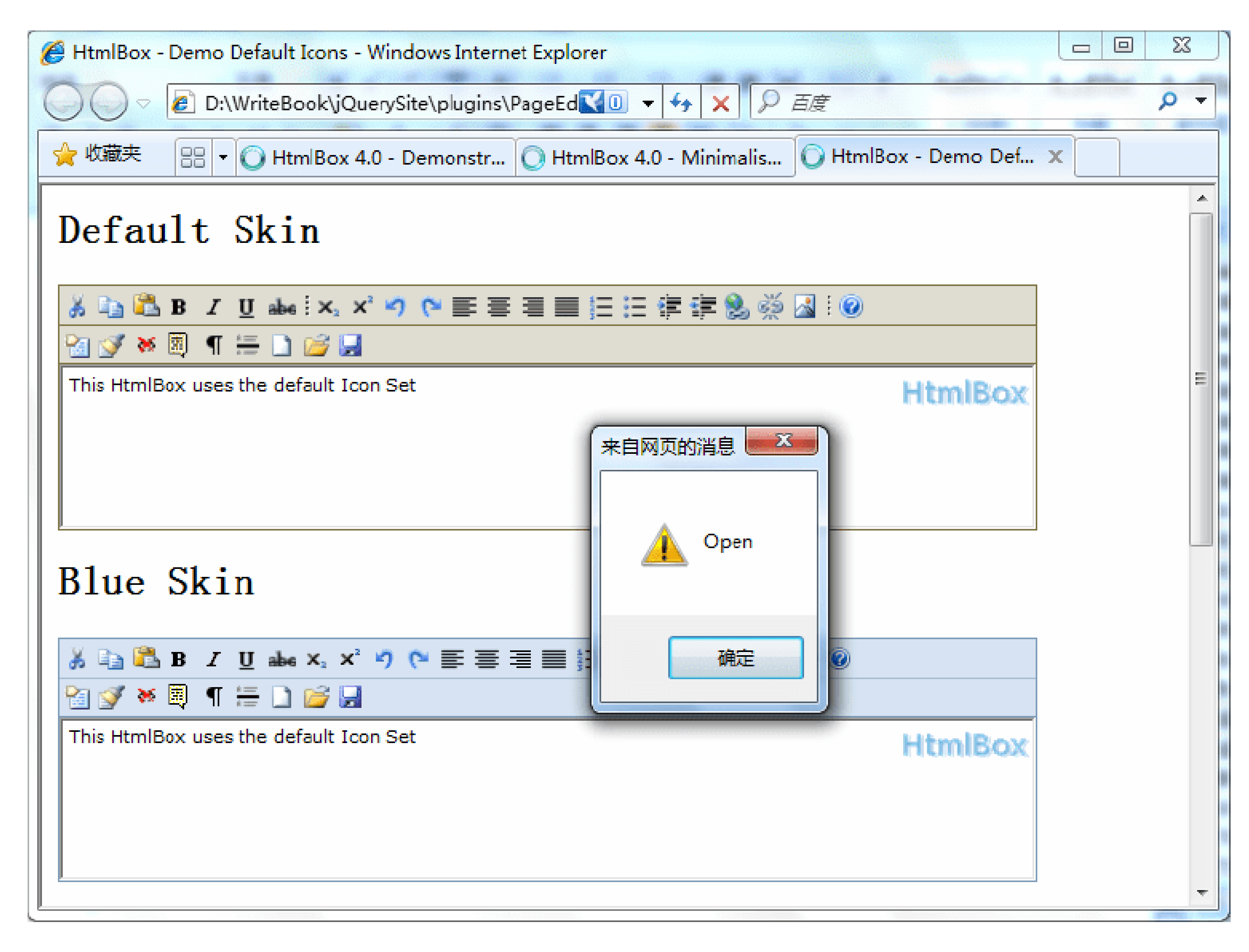


图 11.16　模拟打开功能

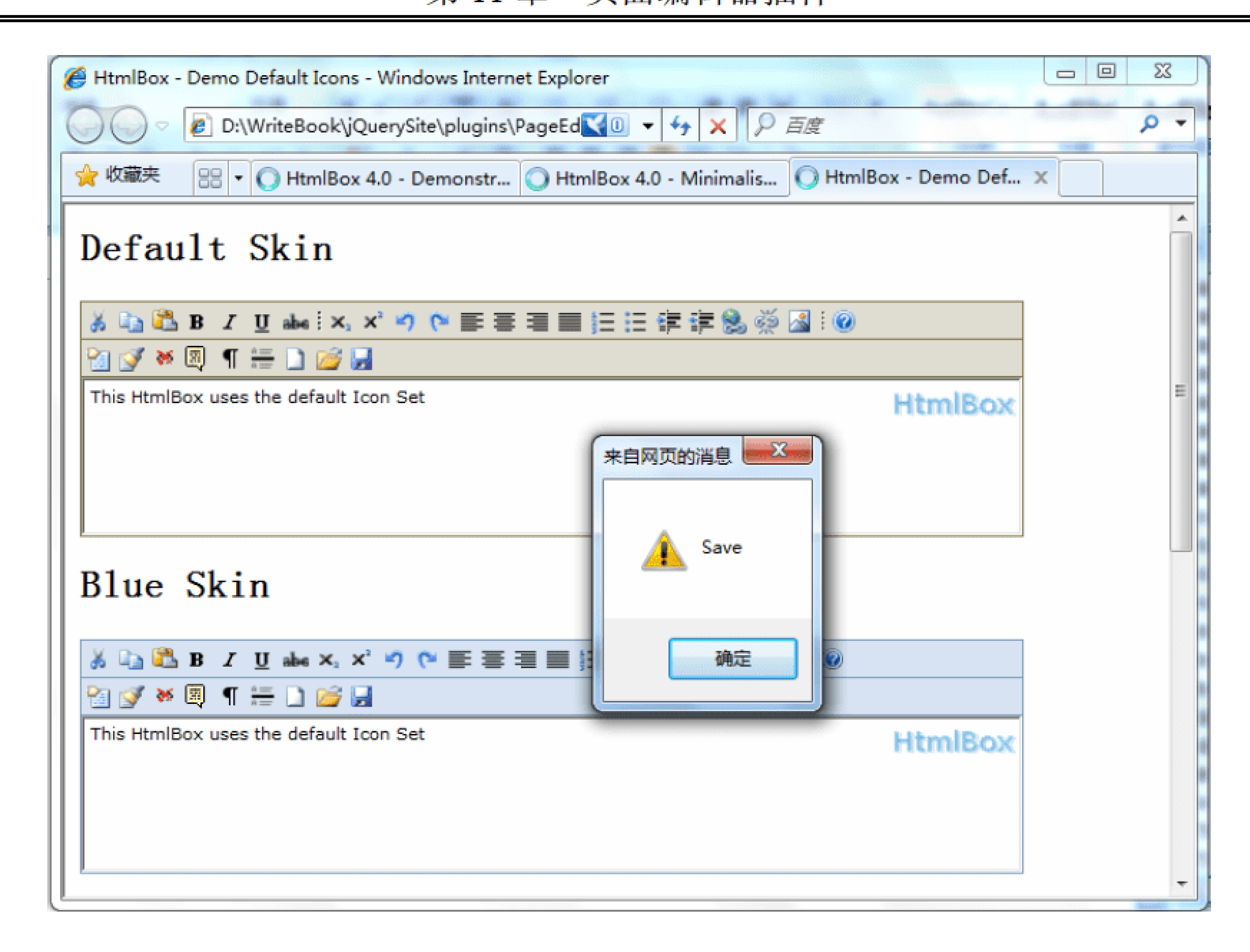


图 11.17　模拟保存功能

11.4　Lightweight RTE 插件

Lightweight RTE 插件也是 jQuery 的富文本编辑器。它也是一个轻量型的插件，易于扩展，为不同的设计模式和资源模式设定不同的工具栏。它的英文介绍网址为 http://code.google.com/p/lwrte/。下面首先介绍一下这个插件的相关函数及说明，如表 11.9 所示。

表 11.9　Lightweight RTE函数说明

函　　数	说　　明
get_content()	返回当前编辑器内容
set_content(content)	设置内容到编辑器
get_select_text()	获取选中文本，不包含标记内容，只工作在设计模式
get_select_html()	获取选中文本，包含标记内容，只工作在设计模式
selection_replace_with(html)	用参数替换选中内容，只工作在设计模式
editor_cmd(command,args)	在设计模式下执行内建命令，携带参数，工作在设计模式
enable_design_mode()	启用设计模式
disable_design_mode(submit)	关闭设计模式，如果参数为 false，则创建新的文本域，从浮动框架复制内容，删除浮动框架，如果参数为 true，则创建新的隐藏域，并提交浮动框架的文本内容
create_toolbar(controls)	创建工具栏，通过控制工具集合
create_panel(title,width)	创建面板，可以指定标题，宽度参数
get_toolbar()	获得当前活动的工具栏
activate_toolbar(editor,tb)	移除编辑器原有工具栏，插入参数 tb 所代表的工具栏
toolbar_click(obj,control)	工具栏上控制按钮的单击事件，执行按钮所包含的执行代码

续表

函　　数	说　　明
get_selection_range()	返回当前选中内容的范围对象，工作在设计模式
get_selected_element()	返回当前鼠标所在 HTML 标记元素，工作在设计模式
set_selected_controls(node, controls)	标记工具栏中的控制按钮只对设计内容中的被选中的 HTML 的 DOM 节点（node）起效，工作在设计模式

下面看一下这个插件的示例，这里使用了插件初始化函数，并设定了插件的基本样式，加载了插件的基本控制按钮和 HTML 标记处理按钮。JavaScript 代码如下：

```
1 <script type="text/JavaScript" src="jquery.js"></script>
2 <script type="text/JavaScript" src="jquery.rte.js"></script>
                                              //插件核心功能文件
3 <script type="text/JavaScript" src="jquery.rte.tb.js"></script>
                                              //插件附加功能文件
4 <script type="text/JavaScript" src="jquery.ocupload-1.1.4.js"></script>
5 <script type="text/JavaScript">
6   $(document).ready(function() {
7       var arr = $('.rte1').rte({
8           css: ['default.css'],             //默认样式设定
9           controls_rte: rte_toolbar,        //使用工具条
10          controls_html: html_toolbar       //使用 HTML 工具条
11      });
12      $('.rte2').rte({
13          css: ['default.css'],
14          width: 450,
15          height: 200,
16          controls_rte: rte_toolbar,
17          controls_html: html_toolbar
18      }, arr);
19  });
20  </script>
```

上述代码第 2、3 行是插件的库文件。第 7、12 行是插件的初始代码。第 8、13 行是插件的样式设定默认样式。第 14、15 行设定了第二个编辑器的初始大小。第 9、16 行设定了编辑器的工具栏。第 10、17 行设定了编辑器设计模式下的工具栏。效果如图 11.18 所示。

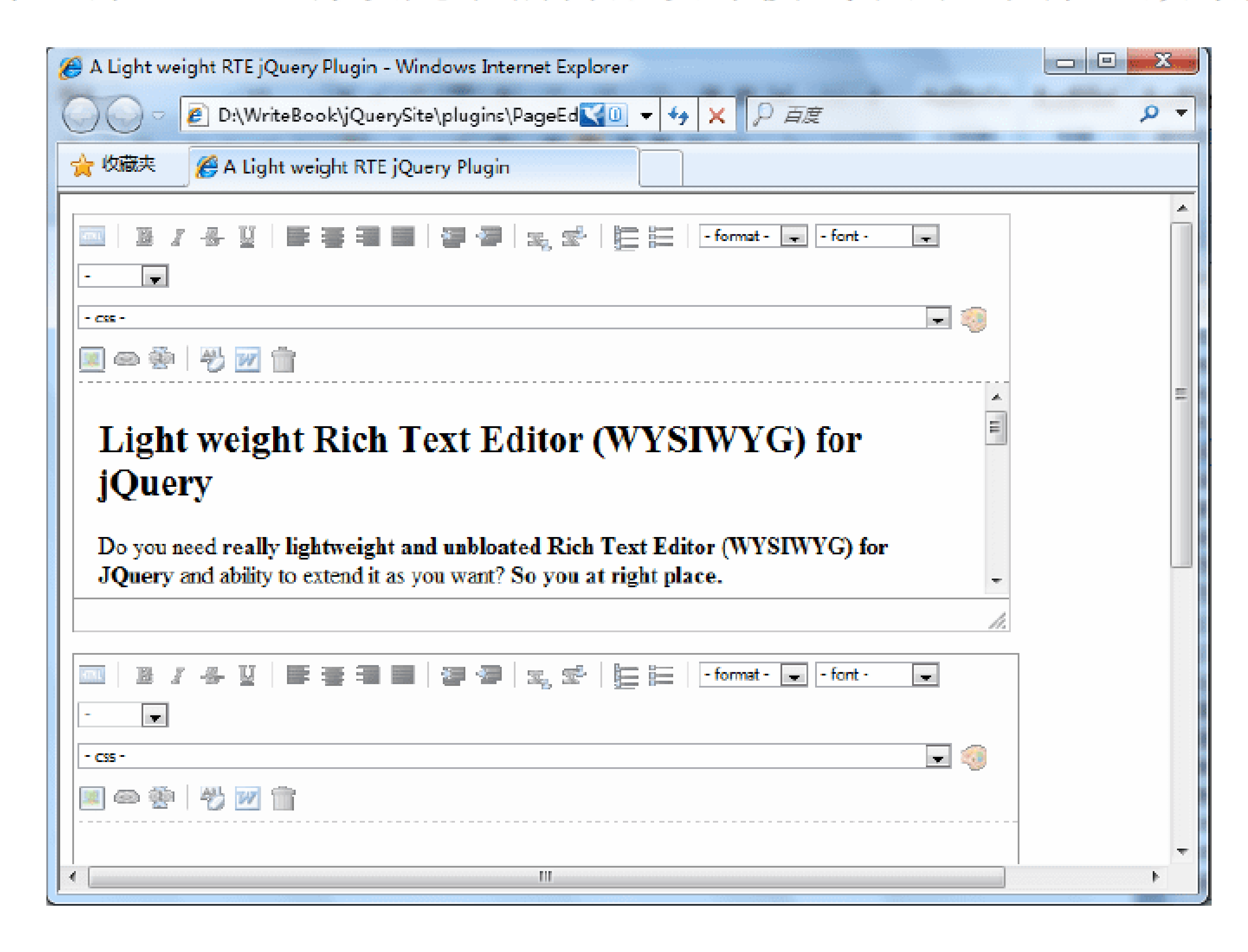


图 11.18　Lightweight RTE 插件效果

11.5　小　　结

文本编辑器是一个良好的用户体验形式。本章主要介绍了文本编辑器插件的使用方法。重点内容是插件的功能与外观定制，同时这部分也是本章的难点。下一章将介绍 jQuery 多媒体插件的使用。

11.6　习　　题

【习题 1】练习 markItUp 插件的使用方法。

【习题 2】练习 jwysiwyg 插件的使用方法。

【习题 3】练习 Lightweight RTE 插件的使用方法。

第 12 章　多　媒　体

在现在的网页中除了文字与静态图片外，还有各种媒体播放形式，如 MP3、Flash、Windows Media Player、Real Player、Silverlight 等。jQuery 也有相应的插件播放这些多媒体文件。多媒体插件除了播放各种媒体文件功能外，还加入了 jQuery 所特有的一些动画特效，更吸引用户。本章将介绍 3 个 jQuery 多媒体插件。

12.1　jQuery.Flash 插件

jQuery.Flash 是基于 jQuery 插件的 JavaScript 代码。它的英文网址是 http://jquery.lukelutman.com/plugins/flash/index.html。该插件主要是为了解决 Flash 在 Web 页面的嵌入式解决方案，与 AC_RunActiveContent、SWFOjbect 功能类似，但是该插件有更多的优点。

（1）完全 W3C 标准。

（2）稳定的交互性。

（3）使用简单，易于扩展。

（4）轻量型，功能强大。

下面通过 4 个示例来介绍这个插件的使用方法。

12.1.1　基本 Flash 文件嵌入

这个示例使用了插件的 src、width、height 属性，分别表示 Flash 文件位置及播放窗口的宽和高。其中，HTML 代码和 CSS 样式设定请参考光盘内容。具体 JavaScript 功能代码如下：

```
1 <script type="text/javascript" src="../../jslib/jquery-1.6.js"></script>
2 <script type="text/javascript" src="JS/jquery.flash.js"></script>
                                                    //添加插件文件
3 <script type="text/javascript">
4   $(document).ready(function(){
5       $('#example').flash({                       //插件初始化
6           src: 'MediaFile/example.swf',           //指定播放的多媒体文件
7           width: 360,                             //插件的显示宽度
8           height: 215                             //插件的显示高度
9       },
10      {
11          version: 8                              //播放文件的版本
12      });
```

```
13  });
14 </script>
```

上述代码第 5 行是插件的初始化函数，第 11 行表示 Flash 版本。效果如图 12.1 所示。

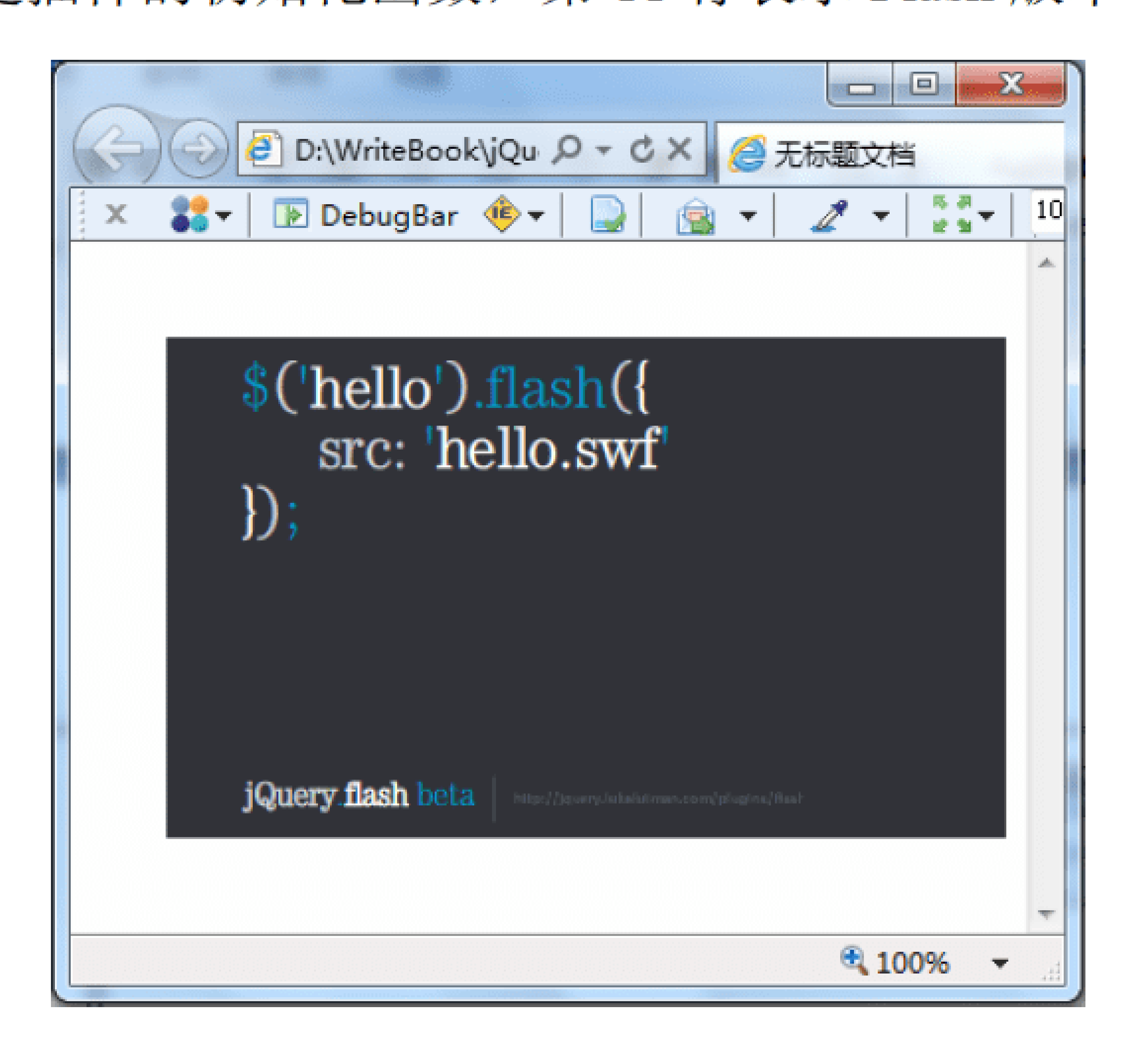


图 12.1　基本 Flash 文件嵌入

12.1.2　Flash 替换文本内容

这个示例中使用 HTML 页面上的文本作为参数，传递 Flash 文件，并在原文本位置播放 Flash。其中，使用到了插件的样式设定参数 src、flashvars 属性、Flash 运行参数。HTML 代码和 CSS 样式设定请参考光盘内容。具体 JavaScript 功能代码如下：

```
1 <script type="text/javascript">
2  $(document).ready(function(){
3      $('h3').flash(
4          {
5              src: 'MediaFile/itc century.swf',
6              flashvars: {
7                  css: [
8                      '* { color: #666666; }',
9                      'a { color: #0055CC; text-decoration: none; }',
10                     'a:hover { text-decoration: underline; }'
11                 ].join(' ')
12             }
13         },
14         { version: 7 },
15         function(htmlOptions) {             //添加文本显示功能
16             htmlOptions.flashvars.txt = this.innerHTML;
17             this.innerHTML = '<span>'+this.innerHTML+'</span>';
18             var $alt = $(this.firstChild);
19             htmlOptions.height = $alt.height();
20             htmlOptions.width = $alt.width();
21             $alt.addClass('alt');
```

```
22                $(this)
23                    .addClass('flash-replaced')
23                    .prepend($.fn.flash.transform(htmlOptions));
25            }
26        );
27  });
28 </script>
```

上述代码第 6 行设定插件的 Flash 参数。第 7～11 行设定文本显示样式。第 14 行设定 Flash 版本。第 15～25 行添加 HTML 文本传入并设定显示样式。其中第 16 行将 h3 内嵌套的内容作为 Flash 参数的文本内容传入插件的 HTML 选项参数。第 17 行向 h3 内嵌套<span>标记。第 18 行获取<span>标记对象，第 19、20 行向 Flash 传入文本所占的高和宽。第 21 行向<span>标记添加 CSS 样式。第 23 行向 h3 添加 CSS 样式。第 24 行向 Flash 传送参数对象。效果如图 12.2 所示。

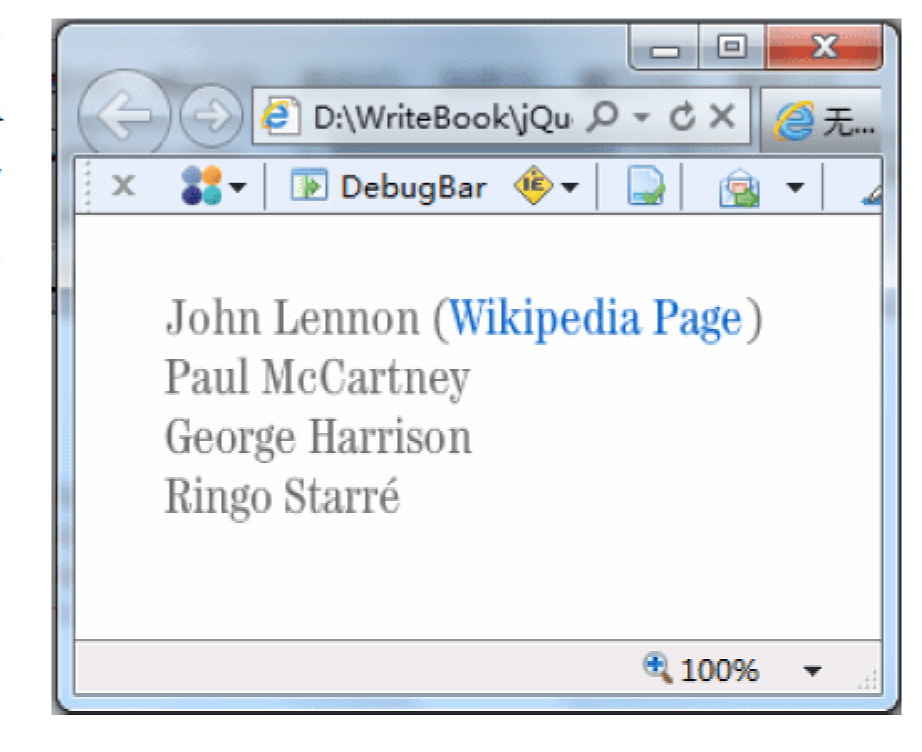


图 12.2　Flash 替换文本内容

12.1.3　MP3 播放示例

这个插件除了可以播放 Flash 文件外，还可以播放 MP3 音频文件。在本示例指定了向插件传入的参数是 MP3 音频文件类型，并指定了文件位置。HTML 代码和 CSS 样式设定请参考光盘内容。具体 JavaScript 功能代码如下：

```
1 <script type="text/javascript" src="../../jslib/jquery.js"></script>
2 <script type="text/javascript" src="JS/jquery.flash.js"></script>
3 <script type="text/javascript">
4   $(document).ready(function(){
5       $('a[@href$="mp3"]').flash(
6           { src: 'MediaFile/singlemp3player.swf', height: 20, width: 100 },
7           { version: 7 },
8           function(htmlOptions) {             //添加超链接功能
9               $this = $(this);
10              htmlOptions.flashvars.file = $this.attr('href');
11              $this.before($.fn.flash.transform(htmlOptions));
12          }
13      );
14  });
15 </script>
```

上述代码第 5 行指定超链接标记中的资源类型为 MP3 音频文件，并调用插件初始化函数。第 6 行指定资源位置，以及播放的高和宽。第 7 行指定 Flash 版本。第 10 行指定向 Flash 传入参数的文件类型参数为超链接的地址属性值。第 11 行将 HTML 参数属性传递给插件。效果如图 12.3 所示。

图 12.3　MP3 播放示例效果

12.1.4　使用内联样式设定插件播放

这个示例在 HTML 标记的 rel 属性内设置插件播放参数，并在插件初始化过程中设定对应参数。HTML 代码和 CSS 样式设定请参考光盘内容。具体 JavaScript 功能代码如下：

```
1 <script type="text/javascript">
2   $(document).ready(function(){
3       $('li').flash(null, { version: 8 }, function(htmlOptions) {
    //通过自定义方法代替静态参数设定播放
4           var $this = $(this);
5           var params = $this.attr('rel').split(':');
6           htmlOptions.src = params[0];            //播放文件位置
7           htmlOptions.width = params[1];          //显示宽度
8           htmlOptions.height = params[2];         //显示高度
9           this.innerHTML = '<div class="alt">'+this.innerHTML+'</div>';
                                                    //内联文本
10          $this.addClass('flash-replaced').prepend($.fn.flash.transform
            (htmlOptions));                         //添加样式和播放参数
11      });
12  });
13 </script>
```

上述代码第 3 行指定在列表项上加载插件并初始化，对于插件样式并没有设置，直接指定 Flash 版本。第 4 行获取列表项对象。第 5 行将列表项中 rel 属性中的值分隔。第 6 行设定播放 Flash 文件位置。第 7、8 行设定插件的宽和高。第 10 行向插件传递参数。效果如图 12.4 和图 12.5 所示。

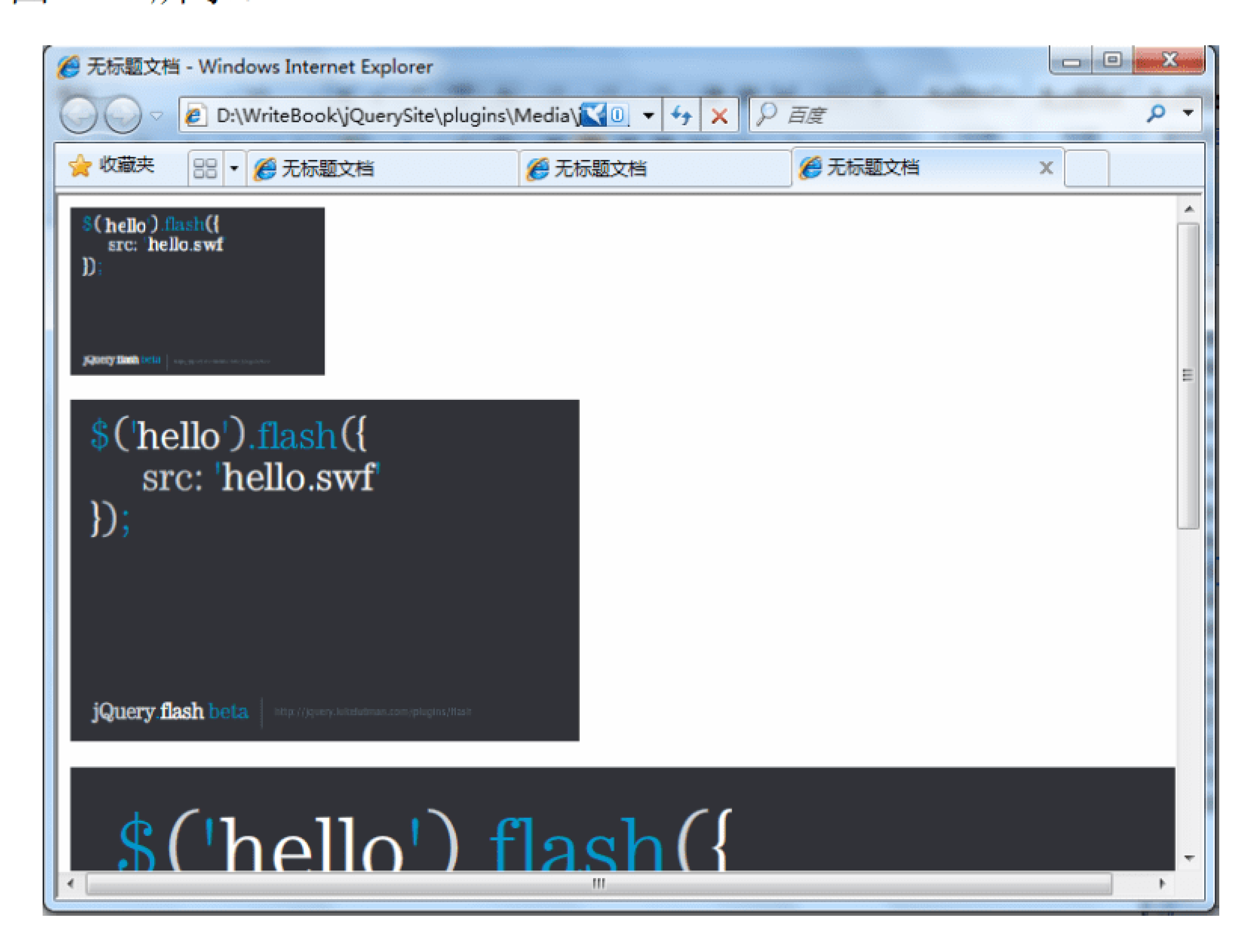


图 12.4　内联样式播放一

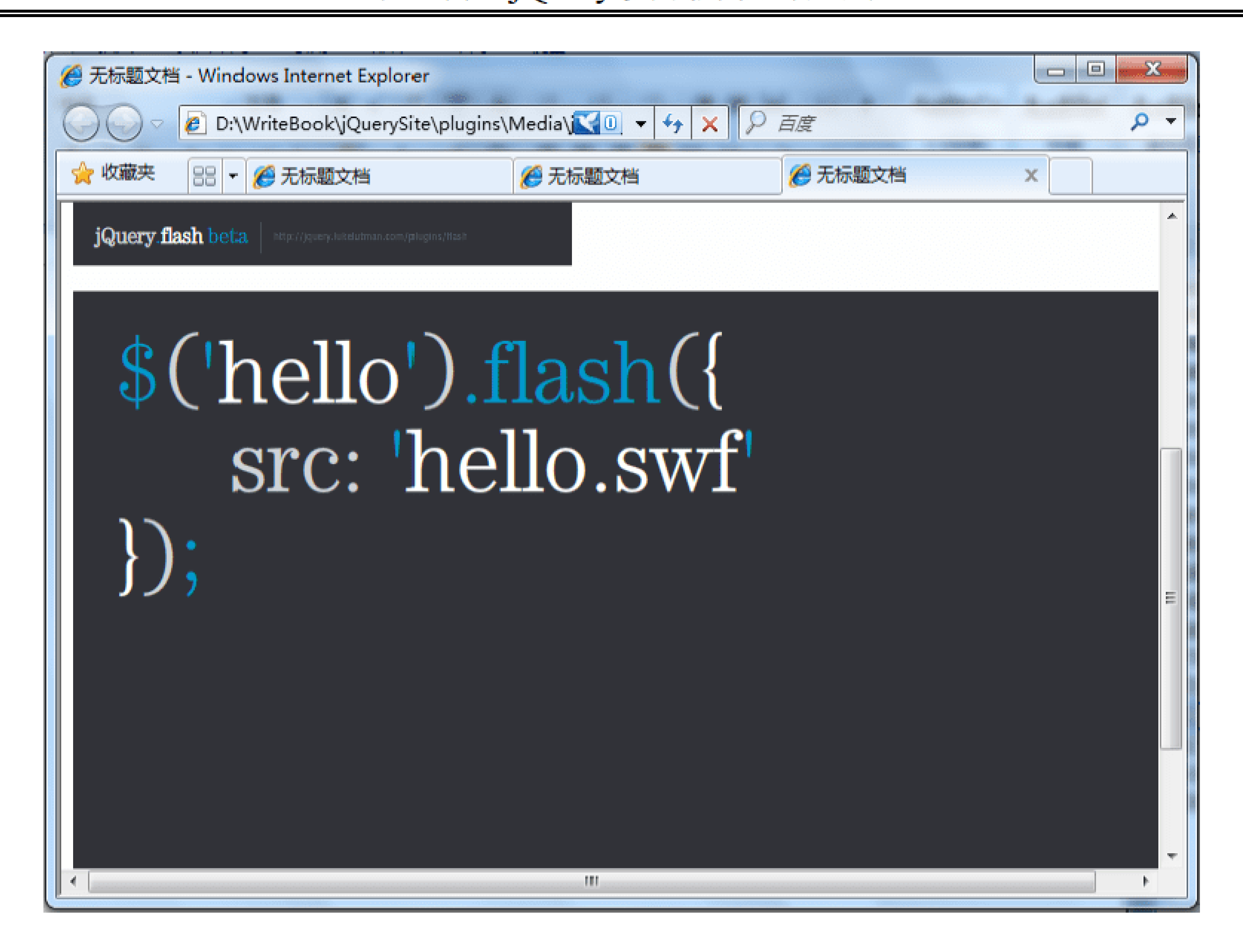


图 12.5　内联样式播放二

12.2　jPlayer 插件

jPlayer 插件和前面讲的 jQuery.Flash 插件类似，也是一个媒体播放插件。因为这个插件的最新版是基于 HTML 5 标准的，因此我们将 IE 8 升级为 IE 9 来查看插件效果。这个插件的英文网址为 http://www.jplayer.org/。下面看一下它的特点。

（1）易于使用和发布。

（2）方便通过 HTML 和 CSS 定制插件样式。

（3）轻量型插件。

（4）免费，开源。

（5）多种解码，跨平台。

12.2.1　jPlayer 插件基本介绍

在前面介绍其他插件时，很多插件在初始化时都有相应属性选项设定。同样，这个插件也有相应的属性选项定义，如表 12.1 所示。

表 12.1　jPlayer插件属性说明

属　　性	类型	默　认　值	说　　明
ready	函数		指定加载的资源文件，以备插件播放
swfPath	字符串	js	指定插件需要播放的 Flash 文件的提取路径，可以根据需要更改为其他路径
solution	字符串	"html, flash"	定义解决方式的主次顺序，可以按照要求颠倒顺序
supplied	字符串	"mp3"	提供给插件的文件格式

续表

属　　性	类型	默　认　值	说　　明
preload	字符串	"metadata"	预加载，可以为"none", "metadata" 和 "auto"
volume	数字	0.8	定义初始化播放音量，从 0 至 1
muted	布尔	false	定义初始化静音状态
backgroundColor	字符串	"#000000"	背景色
cssSelectorAncestor	字符串	"#jp_interface_1"	定义所有 CSS 样式选择器的上层选择器，通常使用 ID 选择器
cssSelector	对象	{cssSelectors}	插件使用的 CSS 选择器对象，有默认的一组 CSS 选择器
idPrefix	字符串	"jp"	在插件内使用的 HTML 元素 ID 的前缀
errorAlerts	布尔	false	是否使用错误报告
warningAlerts	布尔	false	是否使用警告报告
eventType	函数	undefined	负责绑定处理函数的事件类型，参考表 12.2

表 12.2　jPlayer插件事件说明

事　　件	类　　型	说　　明
event	对象	标准插件事件属性
event.jPlayer	对象	插件的信息对象
event.jPlayer.error	对象	插件的错误事件对象
event.jPlayer.error.type	字符串	插件的错误事件对象类型
event.jPlayer.error.context	字符串	发生错误的原因
event.jPlayer.error.message	字符串	描述错误的消息
event.jPlayer.error.hint	字符串	修正错误的建议
event.jPlayer.flash	对象	关于 Flash 解决方式的信息
event.jPlayer.html	对象	关于 HTML 解决方式的信息
event.jPlayer.status	对象	插件状态
event.jPlayer.status.src	字符串	在插件中使用的 URL 地址
event.jPlayer.status.file	对象	设置媒体内容时，代表媒体对象文件
event.jPlayer.status.seekPercent	数值	可查找的百分比
event.jPlayer.status.currentPercentRelative	数值	当前查找时间的百分比
event.jPlayer.status.currentPercentAbsolute	数值	当前持续时间百分比
event.jPlayer.status.currentTime	数值	以秒为单位表示当前时间
event.jPlayer.status.duration	数值	多媒体文件的总播放时间
event.jPlayer.status.volume	数值	插件音量
event.jPlayer.status.muted	布尔	插件是否静音
event.jPlayer.status.srcSet	布尔	多媒体资源是否被设置
event.jPlayer.status.paused	布尔	是否暂停
event.jPlayer.status.waitForPlay	布尔	是否等待播放
event.jPlayer.status.waitForLoad	布尔	是否等待加载
event.jPlayer.status.video	布尔	是否是视频
event.jPlayer.status.width	字符串	插件样式的宽
event.jPlayer.status.height	字符串	插件样式的高
event.jPlayer.version	对象	插件版本

续表

事　　件	类　　型	说　　明
event.jPlayer.version.script	字符串	插件使用的 JavaScript 脚本语言版本
event.jPlayer.version.flash	字符串	插件使用的 Flash 版本
event.jPlayer.version.needFlash	字符串	和 JavaScript 兼容的 Flash 版本
event.jPlayer.warning	对象	插件的警告信息对象
event.jPlayer.warning.type	字符串	插件的警告类型
event.jPlayer.warning.context	字符串	警告原因
event.jPlayer.warning.message	字符串	描述警告的消息
event.jPlayer.warning.hint	字符串	避免警告的建议

插件内置了相关方法供我们调用来操作播放过程，如表 12.3 所示。

表 12.3　jPlayer插件方法说明

方 法 形 式	说　　明
$(id).jPlayer("setMedia", Object: media)	定义播放的多媒体文件，这个方法需要放在其他方法之前，包含的多媒体类型有：mp3、m4a、m4v、oga、ogv、webma、webmv、wav、poster
$(id).jPlayer("clearMedia")	清除多媒体，停止播放
$(id).jPlayer("load")	在播放前预加载多媒体文件
$(id).jPlayer("play", [Number: time])	播放多媒体，参数 time 表示从指定位置播放，时间单位为秒
$(id).jPlayer("pause", [Number: time])	暂停多媒体播放，参数 time 表示在指定位置暂停，时间单位为秒
$(id).jPlayer("pauseOthers")	暂停其他的多媒体播放
$(id).jPlayer("stop")	停止多媒体播放
$(id).jPlayer("playHead", Number: percentOfSeekable)	移动播放位置到进度条的指定位置
$(id).jPlayer("volume", Number: ratio)	媒体播放音量
$(id).jPlayer("mute")	静音操作
$(id).jPlayer("unmute")	取消静音操作
$(id).jPlayer("option", [String: key, [Variable: value]])	设置插件配置信息
$(id).jPlayer("destroy")	移除插件

下面通过示例来介绍这个插件的使用方法。

12.2.2　jPlayer 插件基本使用方式：播放音频文件

这个示例使用了插件的基本事件和方法，包括一些基本属性的使用。首先，需要创建 HTML 静态页面。

```
1       <div class="jp-audio">
2           <div class="jp-type-single">
```

```
3                <div id="jp interface 1" class="jp-interface">
4                    <ul class="jp-controls">
5                        <li><a href="#" class="jp-play" tabindex="1">play
                         </a></li>
6                        <li><a href="#" class="jp-pause" tabindex="1">pause
                         </a></li>
7                        <li><a href="#" class="jp-stop" tabindex="1">stop
                         </a></li>
8                        <li><a href="#" class="jp-mute" tabindex="1">mute
                         </a></li>
9                        <li><a href="#" class="jp-unmute" tabindex="1">unmute
                         </a></li>
10                   </ul>
11                   <div class="jp-progress">
12                       <div class="jp-seek-bar">
13                           <div class="jp-play-bar"></div>
14                       </div>
15                   </div>
16                   <div class="jp-volume-bar">
17                       <div class="jp-volume-bar-value"></div>
18                   </div>
19                   <div class="jp-current-time"></div>
20                   <div class="jp-duration"></div>
21               </div>
22               <div id="jp playlist 1" class="jp-playlist">
23                   <ul>
24                       <li>Bubble</li>
25                   </ul>
26               </div>
27           </div>
28       </div>
```

上述代码中使用了插件的内置 CSS 样式选择器，“jp-type-single”表示单一控件样式，“jp-interface”表示插件界面样式，“jp-controls”表示插件的控制面板样式，“jp-play”表示插件播放按钮样式，“jp-pause”表示插件暂停按钮样式，“jp-stop”表示插件停止按钮样式，“jp-mute”表示静音按钮样式，“jp-unmute”表示取消静音按钮样式，“jp-progress”表示插件播放进度底层 DIV 样式。

“jp-seek-bar”表示播放定位栏样式，“jp-play-bar”表示播放栏样式，“jp-volume-bar”表示音量显示样式，“jp-volume-bar-value”表示当前音量值显示样式，“jp-current-time”表示当前播放时间显示样式，“jp-duration”表示完整播放时间显示样式，“jp-playlist”表示播放曲目列表显示样式。

接下来看一下实现播放功能的 JavaScript 代码：

```
1 <link href="skin/jplayer.blue.monday.css" rel="stylesheet" type=
  "text/css" />
2 <script src="MediaFiles/html5.js"></script>
3 <script type="text/javascript" src="../../../jslib/jquery-1.6.min.js">
  </script>
4 <script type="text/javascript" src="js/jquery.jplayer.min.js"></script>
5 <script type="text/javascript">
```

```
//<![CDATA[
6   $(document).ready(function(){
7       $("#jquery jplayer 1").jPlayer({
8           ready: function () {
9               $(this).jPlayer("setMedia", {            //设置多媒体播放参数
10                  m4a: "MediaFiles/Miaow-07-Bubble.m4a", //m4a 文件
11                  oga: "MediaFiles/Miaow-07-Bubble.ogg"  //oga 文件
12              }).jPlayer("play");                        //调用播放动作
13          },
14          ended: function (event) {                      //结束事件
15              $(this).jPlayer("play");
16          },
17          swfPath: "js",
18          supplied: "m4a, oga"
19      });
20  });
    //]]>
21 </script>
```

上述代码第 7 行调用插件初始化函数。第 8 行使用了插件的准备事件。第 9～12 行设定了向插件中添加多媒体文件，并调用插件的播放方法。第 14、15 行使用了插件的结束事件，同样调用了插件的播放方法。第 17 行指定插件需要使用的 Flash 文件路径。第 18 行指定了插件使用的多媒体文件类型。效果如图 12.6 所示。

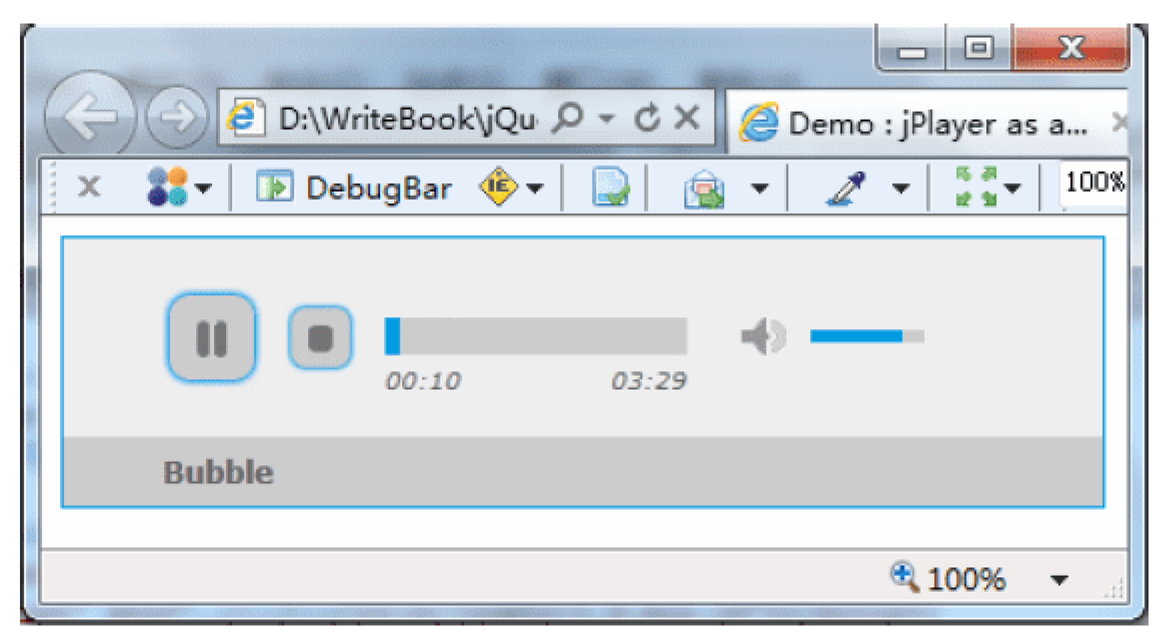


图 12.6　插件的基本使用方式（音频）示例效果

12.2.3　jPlayer 插件基本使用方式：播放视频文件

这个示例使用了插件的基本事件和方法，包括一些基本属性的使用。这个示例的静态页面与上例子类似，在此不过多讲解。不同于上面的例子的是，这里是一个视频文件的播放。下面看一下 JavaScript 功能代码：

```
1 <script type="text/javascript">
  //<![CDATA[
2   $(document).ready(function(){
3       $("#jquery_jplayer_1").jPlayer({
4           ready: function () {
5               $(this).jPlayer("setMedia", {
6                   m4v: "MediaFiles/Big_Buck_Bunny_Trailer_480x270_
                    h264aac.m4v",
7                   ogv: "MediaFiles/Big_Buck_Bunny_Trailer_480x270.ogv",
8                   poster: "MediaFiles/Big_Buck_Bunny_Trailer_480x270.
                    png"
9               });
10          },
11          ended: function (event) {
12              $(this).jPlayer("play");
```

```
13              },
14              swfPath: "js",
15              supplied: "m4v, ogv"
16          });
17  });
    //]]>
18 </script>
```

上述代码和前一个例子基本相同，只是在加入多媒体文件时指定了不同文件类型。效果如图 12.7 所示。

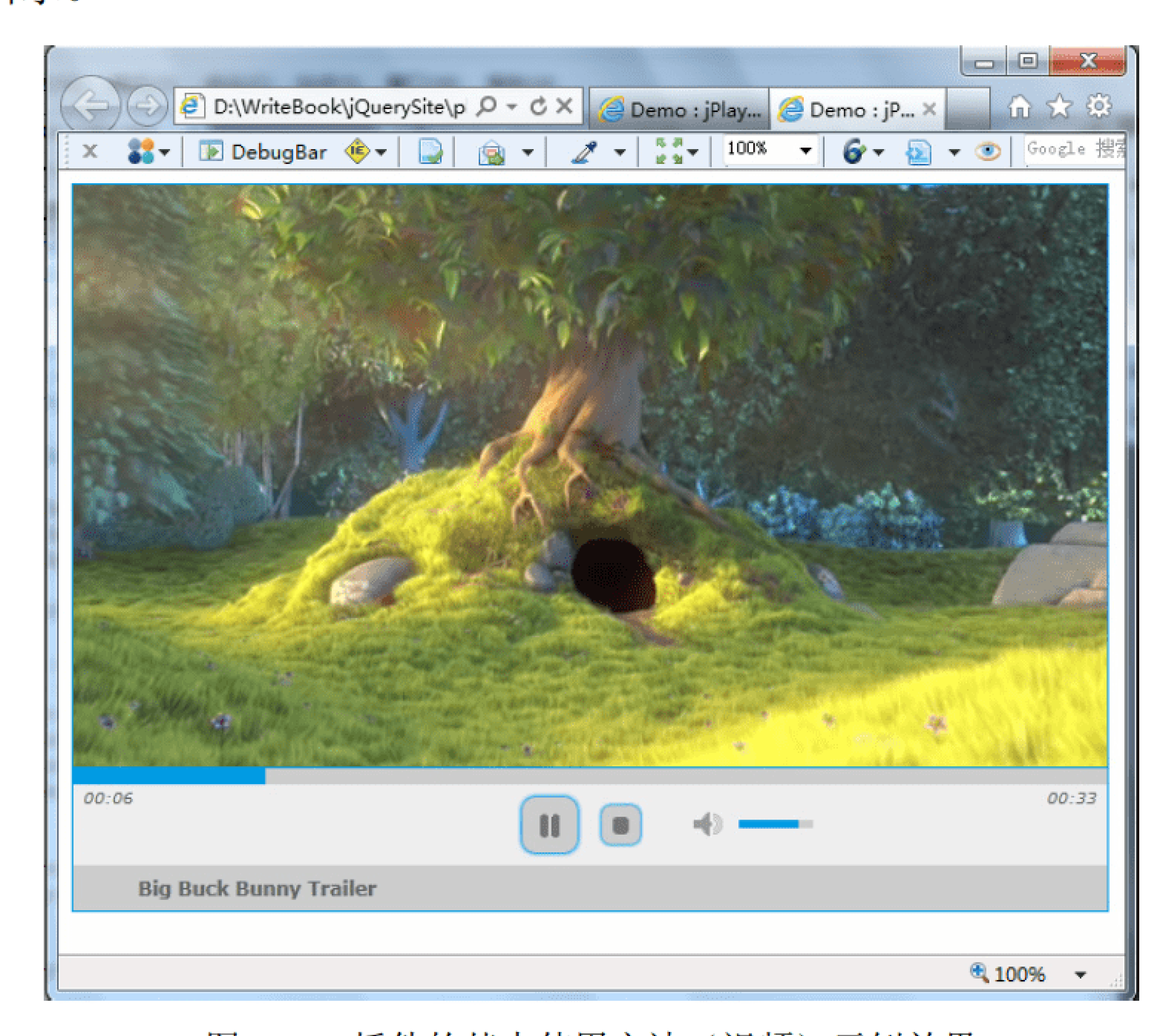


图 12.7　插件的基本使用方法（视频）示例效果

12.2.4　自定义播放器操作

在这个示例中，在页面上添加了两个插件，一个播放视频文件，另一个播放音频文件。每个播放器都加载了多个文件，在播放器的事件操作代码中进行了重新定义。首先，建立 HTML 静态页面：

```
1 <div class="jp-video jp-video-270p">
2           <div class="jp-type-playlist">
3               <div id="jquery jplayer 1" class="jp-jplayer"></div>
4               <div id="jp interface 1" class="jp-interface">
5                   <div class="jp-video-play"></div>
6                   <ul class="jp-controls">
7                       <li><a href="#" class="jp-play" tabindex="1">play
                        </a></li>
8                       <li><a href="#" class="jp-pause" tabindex="1">pause
                        </a></li>
9                       <li><a href="#" class="jp-stop" tabindex="1">stop
```

```
                        </a></li>
10                      <li><a href="#" class="jp-mute" tabindex="1">mute
                        </a></li>
11                      <li><a href="#" class="jp-unmute" tabindex="1">unmute
                        </a></li>
12                      <li><a href="#" class="jp-previous" tabindex="1">
                        previous</a></li>
13                      <li><a href="#" class="jp-next" tabindex="1">next
                        </a></li>
14                  </ul>
15                  <div class="jp-progress">
16                      <div class="jp-seek-bar">
17                          <div class="jp-play-bar"></div>
18                      </div>
19                  </div>
20                  <div class="jp-volume-bar">
21                      <div class="jp-volume-bar-value"></div>
22                  </div>
23                  <div class="jp-current-time"></div>
24                  <div class="jp-duration"></div>
25              </div>
26              <div id="jp playlist 1" class="jp-playlist">
27                  <ul>
28                      <!-- The method Playlist.displayPlaylist() uses
                        this unordered list -->
29                      <li></li>
30                  </ul>
31              </div>
32          </div>
33      </div>
34      <div id="jquery jplayer 2" class="jp-jplayer"></div>
35      <div class="jp-audio">
36          <div class="jp-type-playlist">
37              <div id="jp interface 2" class="jp-interface">
38                  <ul class="jp-controls">
39                      <li><a href="#" class="jp-play" tabindex="1">play
                        </a></li>
40                      <li><a href="#" class="jp-pause" tabindex="1">pause
                        </a></li>
41                      <li><a href="#" class="jp-stop" tabindex="1">stop
                        </a></li>
42                      <li><a href="#" class="jp-mute" tabindex="1">mute
                        </a></li>
43                      <li><a href="#" class="jp-unmute" tabindex="1">unmute
                        </a></li>
44                      <li><a href="#" class="jp-previous" tabindex="1">
                        previous</a></li>
45                      <li><a href="#" class="jp-next" tabindex="1">next
                        </a></li>
46                  </ul>
47                  <div class="jp-progress">
48                      <div class="jp-seek-bar">
```

```
49                          <div class="jp-play-bar"></div>
50                      </div>
51                  </div>
52                  <div class="jp-volume-bar">
53                      <div class="jp-volume-bar-value"></div>
54                  </div>
55                  <div class="jp-current-time"></div>
56                  <div class="jp-duration"></div>
57              </div>
58              <div id="jp playlist 2" class="jp-playlist">
59                  <ul>
60                      <!-- The method Playlist.displayPlaylist() uses
                        this unordered list -->
61                      <li></li>
62                  </ul>
63              </div>
64          </div>
65      </div>
```

上述代码较第一个示例多了两个样式名称，“jp-previous”表示前一个播放文件，“jp-next”表示后一个播放文件。另外，“jp-video-play”表示视频播放插件样式，“jp-audio”表示音频播放插件样式。

前面我们看到的插件示例均是单一文件播放，但是绝大部分情况下播放的多媒体文件是以列表形式出现的，即多个播放文件等待播放。这种功能的设计思路是通过对原有播放插件对象的继续封装，加入新的属性和方法，并加入一定显示样式，使插件从外观到功能上都具有播放多个媒体文件的功能。

实现步骤如下。

（1）创建一个能够容纳播放文件的播放列表对象，我们用这个对象封装了插件对象。

（2）在初始化插件时，指定配置参数。

（3）对播放列表中文件名部分的样式进行设定。

（4）对插件中播放前一文件和播放后一文件等相关功能的按钮编写实现代码。

（5）向播放列表中加入多媒体文件，并完善插件的相关事件功能。

```
1 <script type="text/javascript">
  //<![CDATA[
2   $(document).ready(function(){
3   var Playlist = function(instance, playlist, options) {
                                                //创建播放列表对象原型
4       var self = this;                        //播放列表对象
5       this.instance = instance;               //字符串，指定的播放列表 HTML 元素
6       this.playlist = playlist;               //数组，播放列表
7       this.options = options; //初始化插件时的配置信息
8       this.current = 0;       //当前播放的媒体文件索引，从第一个文件开始播放
9       this.cssId = {          //插件样式，CSS 的 ID 选择值
10          jPlayer: "jquery jplayer ",
11          interface: "jp_interface_",
12          playlist: "jp_playlist_"
13      };
14      this.cssSelector = {};                          //CSS 的选择器
```

```
15        $.each(this.cssId, function(entity, id) {  //创建 ID 选择器字符串
16            self.cssSelector[entity] = "#" + id + self.instance;
17        });
18        if(!this.options.cssSelectorAncestor) {     //创建上级元素的样式选择器
19            this.options.cssSelectorAncestor = this.cssSelector.interface;
20        }
21        $(this.cssSelector.jPlayer).jPlayer(this.options); //初始化播放插件
22        $(this.cssSelector.interface + " .jp-previous").click(function() {
                                                      //播放前一个媒体文件
23            self.playlistPrev();
24            $(this).blur();
25            return false;
26        });
27        $(this.cssSelector.interface + " .jp-next").click(function() {
                                                      //播放后一个媒体文件
28            self.playlistNext();
29            $(this).blur();
30            return false;
31        });
32    };
33    Playlist.prototype = {
34        displayPlaylist: function() {               //定义播放列表显示功能
35            var self = this;
36            $(this.cssSelector.playlist + " ul").empty();  //清空原有列表
37            for (i=0; i < this.playlist.length; i++) {
                                          //根据列表文件个数循环创建列表的 HTML 字符串
38                var listItem = (i === this.playlist.length-1) ? "<li class=
                  'jp-playlist-last'>":"<li>";
39                listItem += "<a href='#' id='" + this.cssId.playlist + this.
                  instance + "_item_" + i +"' tabindex='1'>"+ this.playlist[i].
                  name+"</a>";
40                //创建列表项的链接
41                if(this.playlist[i].free) {     //是否是自由媒体文件
42                    var first = true;
43                    listItem += "<div class='jp-free-media'>(";
44                    $.each(this.playlist[i],  function(property,value)  {
                                                //将不同类型的名称加入到列表中
45                        if($.jPlayer.prototype.format[property]) {
                                                //检查媒体文件类型
46                            if(first) {
47                                first = false;
48                            } else {
49                                listItem += " | ";
50                            }
51                            listItem += "<a id='" + self.cssId.playlist +
                              self.instance + "_item_" + i + "_" + property
                              + "' href='" + value + "' tabindex='1'>" + property
                              + "</a>";
52                        }
53                    });
54                    listItem += ")</span>";
55                }
56                listItem += "</li>";
57                // 显示媒体文件类型
58                $(this.cssSelector.playlist + " ul").append(listItem);
                  //向列表项中加入媒体文件对象
```

```
                //播放列表中媒体文件名链接的单击操作
59              $(this.cssSelector.playlist + "_item_" + i).data("index",
                i).click(function() {
60                  var index = $(this).data("index");
61                  if(self.current !== index) {    //单击的是否是当前播放文件
62                      self.playlistChange(index);//如果不是，则修改播放内容
63                  } else {
64                      $(self.cssSelector.jPlayer).jPlayer("play");
                                                        //直接播放
65                  }
66                  $(this).blur();
67                  return false;
68              });
69              // 强迫使用右键点击播放自由媒体文件
70              if(this.playlist[i].free) {
71                  $.each(this.playlist[i], function(property,value) {
72                      if($.jPlayer.prototype.format[property]) {
                                                    //检查属性是否是媒体文件类型
73                          $(self.cssSelector.playlist + "_item_" + i + "_"
                            + property).data("index", i).click(function() {
74                              var index = $(this).data("index");
75                              $(self.cssSelector.playlist + "_item_" +
                                index).click();
76                              $(this).blur();
77                              return false;
78                          });
79                      }
80                  });
81              }
82          }
83      },
84      playlistInit: function(autoplay) { //播放列表初始化中检测是否是自动播放
85          if(autoplay) {
86              this.playlistChange(this.current);
                                            //如果是自动播放，则从当前文件播放
87          } else {
88              this.playlistConfig(this.current);
                                            //否则配置当前文件为播放文件，但不播放
89          }
90      },
        //下面部分是定义播放列表对象的播放相关功能，包括配置方法、改变播放文件方法、
          播放前一文件，播放后一文件
91      playlistConfig: function(index) {
                                        //配置播放文件，并修改列表中文件名的显示样式
92          $(this.cssSelector.playlist + "_item_" + this.current).
            removeClass("jp-playlist-current").parent().removeClass("jp-
            playlist-current");
93          $(this.cssSelector.playlist + "_item_" + index).addClass("jp-
            playlist-current").parent().addClass("jp-playlist-current");
94          this.current = index;
95          $(this.cssSelector.jPlayer).jPlayer("setMedia", this.playlist
            [this.current]);
96      },
97      playlistChange: function(index) {          //改变播放的媒体文件
98          this.playlistConfig(index);
99          $(this.cssSelector.jPlayer).jPlayer("play");
```

```
100     },
101     playlistNext: function() {                    //播放后一个媒体文件
102         var index = (this.current + 1 < this.playlist.length) ? this.
            current + 1 : 0;
103         this.playlistChange(index);
104     },
105     playlistPrev: function() {                    //播放前一个媒体文件
106         var index = (this.current - 1 >= 0) ? this.current - 1 : this.
            playlist.length - 1;
107         this.playlistChange(index);
108     }
109 };
    //对播放列表添加媒体文件，并对准备播放、结束播放、播放事件功能进行加工，这里添加的
      是视频文件
110 var videoPlaylist = new Playlist("1", [
111     {
112         name:"Big Buck Bunny Trailer",
113         free:true,
114         m4v:"MediaFiles/Big Buck Bunny Trailer 480x270 h264aac.m4v",
115         ogv:"MediaFiles/Big Buck Bunny Trailer 480x270.ogv",
116         poster:"MediaFiles/Big_Buck_Bunny_Trailer_480x270.png"
117     },
118     {
119         name:"Finding Nemo Teaser",
120         m4v: "MediaFiles/Finding_Nemo_Teaser_640x352_h264aac.m4v",
121         ogv: "MediaFiles/Finding Nemo Teaser 640x352.ogv",
122         poster: "MediaFiles/Finding Nemo Teaser 640x352.png"
123     },
124     {
125         name:"Incredibles Teaser",
126         m4v: "MediaFiles/Incredibles Teaser 640x272 h264aac.m4v",
127         ogv: "MediaFiles/Incredibles_Teaser_640x272.ogv",
128         poster: "MediaFiles/Incredibles_Teaser_640x272.png"
129     }
130 ], {
131     ready: function() { //准备播放事件
132         videoPlaylist.displayPlaylist();
133         videoPlaylist.playlistInit(false); // 参数为自动播放的布尔值设置
134     },
135     ended: function() {             //结束播放事件，转为播放下一个媒体文件
136         videoPlaylist.playlistNext();
137     },
138     play: function() {              //播放事件
139         $(this).jPlayer("pauseOthers");     //暂停其他文件播放
140     },
141     swfPath: "js",
142     supplied: "ogv, m4v"
143 });
    //对播放列表添加媒体文件，并对准备播放、结束播放、播放事件功能进行加工，这里添加的
      是音频文件
144 var audioPlaylist = new Playlist("2", [
145     {
146         name:"Tempered Song",
147         mp3:"MediaFiles/Miaow-01-Tempered-song.mp3",
148         oga:"MediaFiles/Miaow-01-Tempered-song.ogg"
149     },
```

```
150     {
151         name:"Hidden",
152         mp3:"MediaFiles/Miaow-02-Hidden.mp3",
153         oga:"MediaFiles/Miaow-02-Hidden.ogg"
154     },
155     {
156         name:"Lentement",
157         free:true,
158         mp3:"MediaFiles/Miaow-03-Lentement.mp3",
159         oga:"MediaFiles/Miaow-03-Lentement.ogg"
160     },
161     {
162         name:"Lismore",
163         free:true,
164         mp3:"MediaFiles/Miaow-04-Lismore.mp3",
165         oga:"MediaFiles/Miaow-04-Lismore.ogg"
166     },
167     {
168         name:"The Separation",
169         mp3:"MediaFiles/Miaow-05-The-separation.mp3",
170         oga:"MediaFiles/Miaow-05-The-separation.ogg"
171     },
172     {
173         name:"Beside Me",
174         mp3:"MediaFiles/Miaow-06-Beside-me.mp3",
175         oga:"MediaFiles/Miaow-06-Beside-me.ogg"
176     },
177     {
178         name:"Bubble",
179         free:true,
180         mp3:"MediaFiles/Miaow-07-Bubble.mp3",
181         oga:"MediaFiles/Miaow-07-Bubble.ogg"
182     },
183     {
184         name:"Stirring of a Fool",
185         mp3:"MediaFiles/Miaow-08-Stirring-of-a-fool.mp3",
186         oga:"MediaFiles/Miaow-08-Stirring-of-a-fool.ogg"
187     },
188     {
189         name:"Partir",
190         mp3:"MediaFiles/Miaow-09-Partir.mp3",
191         oga:"MediaFiles/Miaow-09-Partir.ogg"
192     },
193     {
194         name:"Thin Ice",
195         free:true,
196         mp3:"MediaFiles/Miaow-10-Thin-ice.mp3",
197         oga:"MediaFiles/Miaow-10-Thin-ice.ogg"
198     }
199 ], {
200     ready: function() {
201         audioPlaylist.displayPlaylist();
202         audioPlaylist.playlistInit(false);// Parameter is a boolean for
            autoplay.
203     },
204     ended: function() {
205         audioPlaylist.playlistNext();
```

```
206         },
207         play: function() {
208             $(this).jPlayer("pauseOthers");
209         },
210         swfPath: "js",
211         supplied: "oga, mp3"
212 });
213 });
//]]>
214 </script>
```

上述代码第 3～32 行定义 Playlist 对象形式，其中 instance 表示插件列表对象实例，playlist 表示插件数组，options 表示插件初始化参数。第 8 行设定初始播放顺序为第一个媒体文件；第 9～13 行设定代表插件的 HTML 元素的 ID 匹配内容。第 15～17 行设定每个插件的 CSS 选择器。第 18～20 行获取上层样式设定的 ID。第 21 行利用 options 参数初始化播放器。第 22～26 行设定前一个媒体文件按钮的单击处理事件。第 27～32 行设定后一个媒体文件按钮的单击处理事件。

第 33～109 行设定 Playlist 对象属性及操作。其中第 34～83 行创建播放列表。第 38、39 行创建播放列表上单击播放文件名的超链接。第 41～54 行判断播放文件是否显示多种播放格式（mp3|oga）。第 59～68 行设定每个播放超链接的单击事件，如果单击的不是当前播放文件，则使用播放改变函数处理，否则重新播放当前文件。第 70～81 行判断播放超链接是否具有多种播放格式，强迫通过右键单击访问。

第 84～90 行为播放列表初始化函数，判断是否允许自动播放，如果允许则自动播放当前文件，否则配置播放文件位置。第 91～96 行配置播放列表播放顺序。第 97～100 行改变播放列表顺序。第 101～104 行播放下一个媒体文件。第 105～109 行播放前一个媒体文件。第 110～143 行初始化第一个视频播放插件。第 144～211 行初始化第二个音频播放插件，这两部分代码与前面的例子类似。效果如图 12.8 和图 12.9 所示。

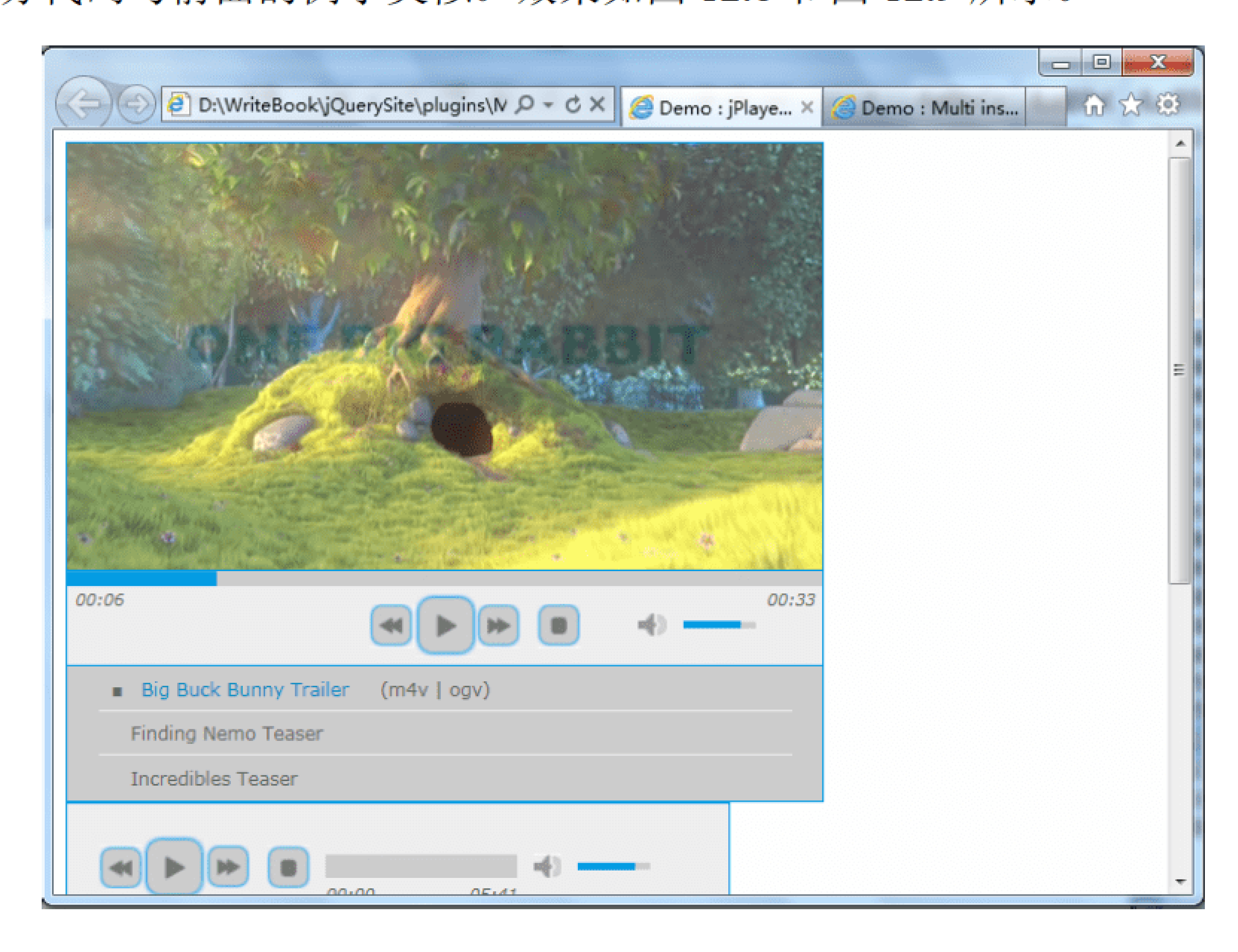


图 12.8　自定义播放器操作视频播放

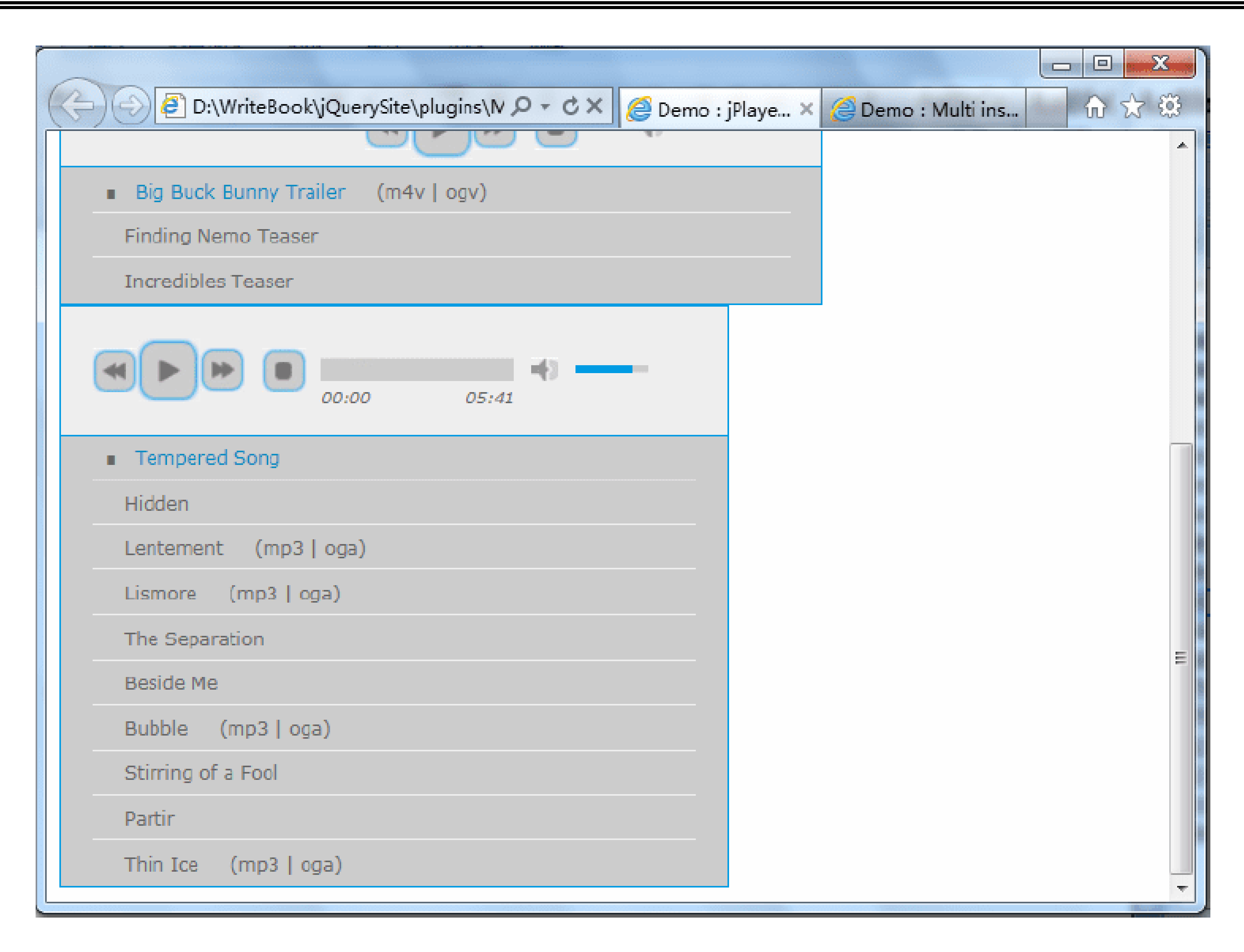


图 12.9　自定义播放器操作音频播放

12.3　jlEmbed 插件

jlEmbed 插件几乎可以嵌入任何一种媒体类型到网页上。它内建了音乐列表支持。这个插件和前面所介绍的插件不同之处在于，它的所有属性设定都写在元素中，而不是通过初始化函数来完成设定。它的英文网址为 http://jasonlau.biz/home/jlembed-for-jquery。

下面看一下这个插件的属性说明，如表 12.4 所示。

表 12.4　jlEmbed插件属性说明

属　　性	类型	默认值	说　　明
autostart	布尔	true	页面加载后自动播放，但是在 adobeflash, musicplayer, silverlight 模式下无效
caption	字符串	null	在嵌入的播放器下加入文本或者 HTML
controls	布尔	false	只在 youtube 模式下使用，在播放器下添加通用控制按钮
debug	布尔	false	只在 youtube 模式下使用，显示嵌入对象的父内容
error_alert	布尔	false	当插件运行发生错误时抛出 JavaScript 的警告对话框
format	字符串		HTML 输出格式
height	数值	varis	插件的高
id	字符串		嵌入对象的 ID
loop	布尔	false	自动循环播放
mode	字符串	null	设置插件模式
params	字符串		自定义参数
screw	布尔	false	当播放器在页面可见时加载
shuffle	布尔	false	重新排列播放列表
src	字符串		媒体资源位置
stop_onerror	布尔	false	只在 youtube 模式下使用，当发生错误时停止
volume	数值	100	只在 youtube 模式下使用，设置默认音量
width	数值	varies	嵌入播放器的高

下面通过示例来了解这个插件如何加载各种媒体文件。

12.3.1　插件基本使用

这个示例使用插件加载一个 Flash 文件。前面介绍过这个插件属于自动初始化，所以我们只需要引入相关的 jQuery 库文件：

```
<script type="text/javascript" src="../../jslib/jquery-1.6.js"></script>
<script type="text/javascript" src="JS/jquery.jlembed.js"></script>
                                                    //插件核心功能文件
```

然后建立静态页面，并在页面的 HTML 标记上将插件需要的属性添加进去就可以了：

```
<div id="mydiv" class="jlembed" data-id="myplayer" data-mode="adobeflash"
data-params='{"wmode":"transparent", "quality":"high", "allowscriptaccess":
"always", "allowfullscreen":"true"}' data-format="swfobject" data-src=
'MediaFile/example.swf' data-width="294" data-height="196" data-caption=
"This is funny." data-screw="true"></div>
```

上述代码实现了 Flash 文件的加载，这里需要说明的是如果希望在元素上内嵌插件，则该元素的类属性需要设定为“jlembed”，后面的示例也是如此。表 12.4 所提到的所有属性如果在标记中需要指定，则要在属性前加上 data 前缀。这个示例使用 params 设定了 Flash 文件加载效果，窗口模式透明（"wmode":"transparent"），画面质量高（"quality":"high"），允许脚本访问（"allowscriptaccess": "always"），允许全屏（"allowfullscreen":"true"）。效果如图 12.10 所示。

12.3.2　使用 Windows Media Player 播放

这个示例使用了 Windows Media Player 播放文件，代码如下：

```
<div id="mydiv" class="jlembed" data-id="myplayer" data-mode="windowsmedia"
data-params='{"console":"cons","controls":"all"}' data-format="objectembed"
data-src="MediaFile/mixdown.mp3"    data-width="300"    data-height="40"
data-autostart="false" data-loop="true" data-caption="This is my song."
data-screw="true"></div>
```

上述代码指定了播放 MP3 文件的格式，并使用 Windows Media Player 软件来播放。如果你的计算机中也安装了 Real Player，则将 data-mode= "windowsmedia"修改成 data-mode="realplayer"就可以使用 Real Player 来播放了。效果如图 12.11 所示。

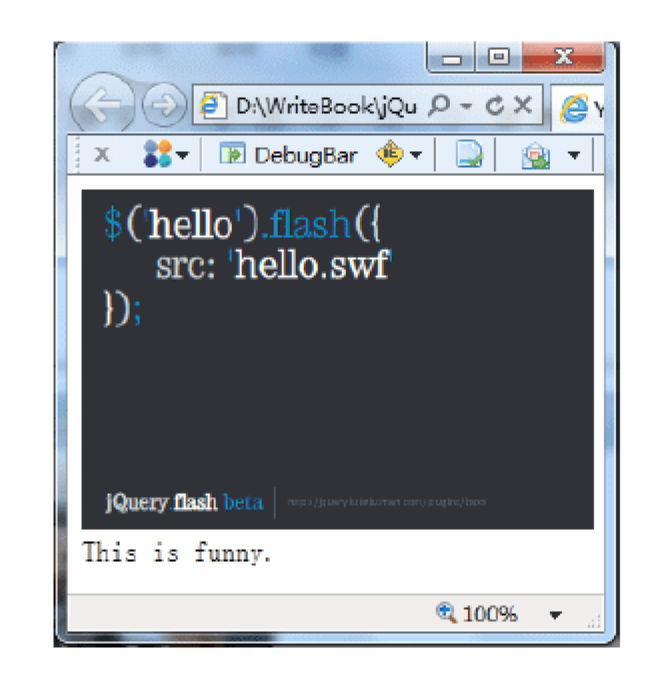


图 12.10　jlEmbed 基本使用示例效果

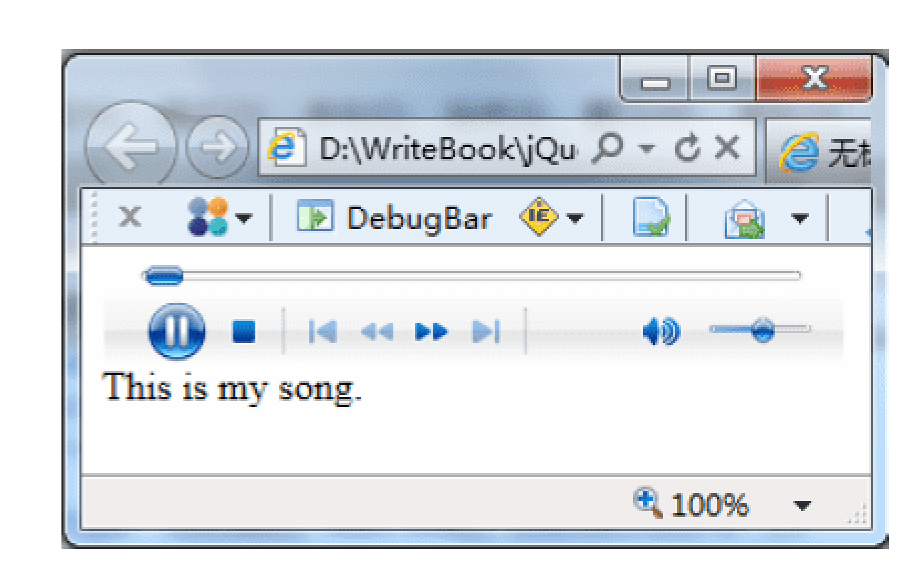


图 12.11　使用 Windows Media Player 播放 MP3

12.3.3　使用 QuickTime 播放

这个示例使用 QuickTime 播放多媒体文件。代码如下：

```
<div id="mydiv" class="jlembed" data-id="myplayer" data-mode="quicktime"
data-params='{"controller":"true"}' data-format="objectembed" data-src=
'{"Mixdown1":"MediaFile/mixdown.mp3","Mixdown2":"MediaFile/mixdown.mp3"
}' data-width="300" data-height="45" data-autostart="false" data-loop=
"true" data-shuffle="false" data-caption="This is my song." data-screw=
"true"></div>
```

上述代码指定了使用 QuickTime 播放 MP3 多媒体资源文件，效果如图 12.12 所示。

12.3.4　使用 SilverLight 播放

这个示例使用 SilverLight 播放多媒体文件。代码如下：

```
<div id="mydiv" class="jlembed" data-id="myplayer" data-mode="silverlight"
data-params='{"onLoad":"  slLoad0", "onError":"  slError0", "windowless":
"false", "maxFramerate":"30"}' data-format="objectembed" data-src='MediaFile/
WeatherWidget.xap' data-width="300" data-height="250" data-caption="This
is a test caption." data-screw="true"></div>
```

上述代码对 SilverLight 的使用做了参数设定，加载事件（"onLoad":"__slLoad0"），发生错误事件（"onError":"__slError0"），不显示无窗口插件（"windowless":"false"），设置每秒可呈现的最大帧数（"maxFramerate":"30"）。效果如图 12.13 所示。

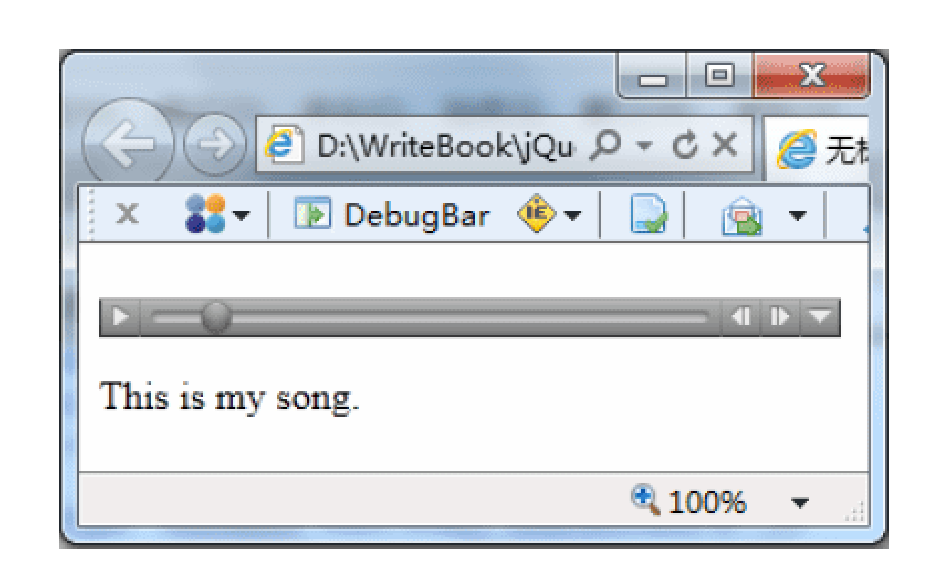


图 12.12　使用 QuickTime 播放 MP3

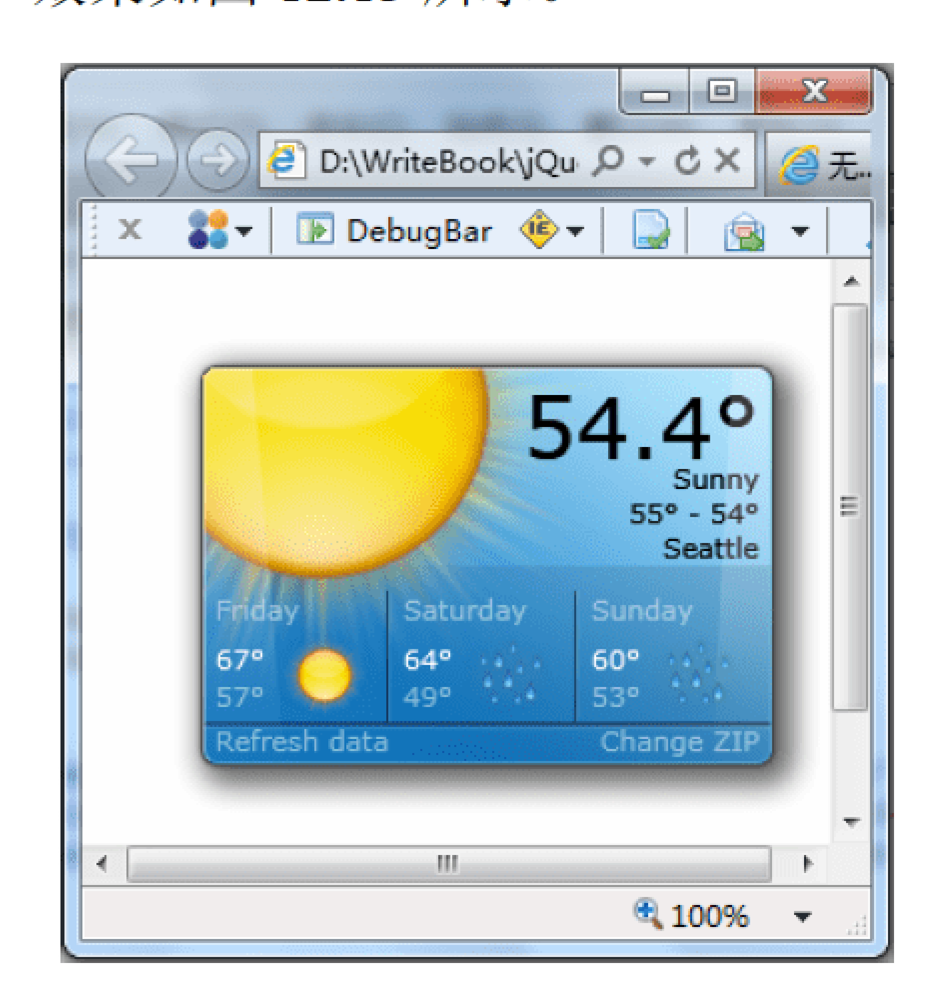


图 12.13　利用 SilverLight 播放多媒体文件

12.3.5　使用 Flash Player 播放

这个示例调用了 Adobe 的 Flash Player 播放器播放 Flash 文件，代码如下：

```
<div id="mydiv" class="jlembed" data-id="myplayer" data-mode="adobeflash"
data-params='{"wmode":"transparent", "quality":"high", "allowscriptaccess":
"always", "allowfullscreen":"true"}' data-format="swfobject" data-src=
'MediaFile/example.swf' data-width="294" data-height="196" data-caption=
"This is funny." data-screw="true"></div>
```

上述代码与 12.3.1 节介绍的示例基本相同，效果如图 12.14 所示。



图 12.14　利用 Adobe Flash Player 播放 Flash 文件

12.4　小　　结

多媒体播放插件能够帮助界面设计人员更灵活地表现页面内容，也能让用户看到更生动的页面。本章主要介绍了多媒体插件的使用方法。重点内容是插件的功能与外观定制，同时这部分也是本章的难点。下一章将介绍 jQuery 动画设计。

12.5　习　　题

【习题 1】练习 jQuery.Flash 插件的使用方法。

【习题 2】练习 jPlayer 插件的使用方法。

【习题 3】练习 jlEmbed 插件的使用方法。

第 13 章　动 画 设 计

动画可以更直观、更生动地表现出设计者的意图。动画可以通过专业的动画软件如 Flash 等来制作。现在网页上动画所占的比例越来越大。jQuery 除了可以实现前面章节所完成的效果外，还可以在一定程度上实现动画效果。本章讲解如何利用 jQuery 在网页上制作动画。

13.1　jQuery 动画基础

本节将介绍利用 jQuery 实现动画效果的基础知识。在前面的章节中介绍了 jQuery 制作动画的一些相关函数，本节总结一下这些函数，并通过示例来说明应用。

13.1.1　jQuery 动画函数

jQuery 的动画函数总共分成 4 类。

（1）基本动画函数：既有透明度渐变，又有滑动效果，是最常用的动画效果函数。

（2）滑动动画函数：仅适用滑动渐变动画效果。

（3）淡入淡出动画函数：仅适用透明度渐变动画效果。

（4）自定义动画函数：作为上述三种动画函数的补充和扩展。

下面将这几种动画函数总结一下。首先是基本动画函数，如表 13.1 所示。

表 13.1　jQuery基本动画函数总结

函　　数	使 用 方 式	说　　明
show()	$("p").show()	显示隐藏的匹配元素。 这个函数就是 'show(speed, [callback])'的无动画版本。如果选择的元素是可见的，这个方法将不会改变任何东西。无论这个元素是通过 hide()方法隐藏的还是在 CSS 里设置了 display:none;，这个方法都将有效
show(speed,[callback])	$("p").show("slow"); $("p").show("fast",function(){ 　$(this).text("Animation Done!"); 　});	以优雅的动画显示所有匹配的元素，并在显示完成后可选地触发一个回调函数。可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅
hide()	$("p").hide()	隐藏显示的元素。这个函数就是'hide(speed, [callback])' 的无动画版本。如果选择的元素是隐藏的，这个方法将不会改变任何东西

续表

函　　数	使 用 方 式	说　　明

hide(speed,[callback])	$("p").hide("slow");
	$("p").hide("fast",function(){
	　alert("Animation Done.");
	});
		以优雅的动画隐藏所有匹配的元素，并在显示完成后可选地触发一个回调函数。可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅

toggle()	$("p").toggle()	切换元素的可见状态。如果元素是可见的，切换为隐藏的；如果元素是隐藏的，切换为可见的

toggle(switch)	var flip = 0;
	　$("button").click(function () {
	　　$("p").toggle(flip++ % 2 == 0);
	　});
		根据 switch 参数切换元素的可见状态（true 为可见，false 为隐藏）。如果 switch 设为 true，则调用 show()方法来显示匹配的元素，如果 switch 设为 false，则调用 hide()来隐藏元素

toggle(speed,[callback])	$("p").toggle("slow");
	$("p").toggle("fast",function(){
	　alert("Animation Done.");
	});
		以优雅的动画切换所有匹配的元素，并在显示完成后可选地触发一个回调函数。可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅

滑动动画函数如表 13.2 所示。

表 13.2　jQuery滑动动画函数总结

函　　数	使 用 方 式	说　　明

slideDown(speed,[Callback])	$("p").slideDown("slow");
	$("p").slideDown("fast",function(){
	　alert("Animation Done.");
	});
		通过高度变化（向下增大）来动态地显示所有匹配的元素，在显示完成后可选地触发一个回调函数。这个动画效果只调整元素的高度，可以使匹配的元素以“滑动”的方式显示出来。在 jQuery 1.3 中，上下的 padding 和 margin 也会有动画，效果更流畅

slideUp(speed,[Callback])	$("p").slideUp("slow");
	$("p").slideUp("fast",function(){
	　alert("Animation Done.");
	});
		通过高度变化（向上减小）来动态地隐藏所有匹配的元素，在隐藏完成后可选地触发一个回调函数

slideToogle(speed,[Callback])	$("p").slideToggle("slow");
	$("p").slideToggle("fast",function(){
	　alert("Animation Done.");
	});
		通过高度变化来切换所有匹配元素的可见性，并在切换完成后可选地触发一个回调函数

淡入淡出动画函数如表 13.3 所示。

表 13.3　jQuery淡入淡出动画函数总结

函　　数	使 用 方 式	说　　明

fadeIn(speed,[Callback])	$("p").fadeIn("slow");
	$("p").fadeIn("fast",function(){
	　alert("Animation Done.");
	});
		通过不透明度的变化来实现所有匹配元素的淡入效果，并在动画完成后可选地触发一个回调函数。这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化

续表

函　　数	使 用 方 式	说　　明

fadeOut(speed,[Callback])	$("p").fadeOut("slow");
	$("p").fadeOut("fast",function(){
	　alert("Animation Done.");
	});
		通过不透明度的变化来实现所有匹配元素的淡出效果，并在动画完成后可选地触发一个回调函数

fadeTo(speed,[Callback])	$("p").fadeTo("slow", 0.66);
	$("p").fadeTo("fast", 0.25, function()
	{
	　alert("Animation Done.");
	});
		把所有匹配元素的不透明度以渐进方式调整到指定的不透明度，并在动画完成后可选地触发一个回调函数

自定义动画函数如表 13.4 所示。

表 13.4　jQuery自定义动画函数总结

函　　数	使 用 方 式	总　　结
animate(param, [dur],[e],[fn])	$("p").animate({ 　height: 'toggle', opacity: 'toggle' }, "slow"); $("p").animate({ 　opacity: 'show' }, "slow", "easein");	指定动画形式及结果样式属性对象。每个属性的值表示这个样式属性到多少时动画结束。如果是一个数值，样式属性就会从当前的值渐变到指定的值。如果使用的是“hide”、“show”或“toggle”这样的字符串值，则会为该属性调用默认的动画形式。param，一组包含作为动画属性和终值的样式属性以及其值的集合；duration，三种预定速度之一的字符串（“slow”、“normal”or“fast”）或表示动画时长的毫秒数值（如 1000）；easing，要使用的擦除效果的名称（需要插件支持），jQuery 默认提供“linear”和“swing”；callback，在动画完成时执行的函数
animate (param,options)	$("p").animate({ 　left: 50, opacity: 'show' }, { duration: 500 }); $("p").animate({ 　opacity: 'show' }, { duration: "slow", easing: "easein" });	指定动画形式及结果样式属性对象。每个属性的值表示这个样式属性到多少时动画结束。如果是一个数值，样式属性就会从当前的值渐变到指定的值。如果使用的是“hide”、“show”或“toggle”这样的字符串值，则会为该属性调用默认的动画形式。param，一组包含作为动画属性和终值的样式属性以及其值的集合；options，一组包含动画选项的值的集合；duration，三种预定速度之一的字符串（“slow”、“normal”or“fast”）或表示动画时长的毫秒数值（如 1000）；easing，要使用的擦除效果的名称（需要插件支持），jQuery 默认提供“linear”和“swing”；complete，在动画完成时执行的函数；step，动画步进时执行的函数；queue，（默认值：true）设定为 false 将使此动画不进入动画队列
stop([clearQueue], [gotoEnd])	$(".block").stop();	停止所有在指定元素上正在运行的动画。如果队列中有等待执行的动画（并且 clearQueue 没有设为 true），它们将被马上执行。clearQueue，如果设置成 true，则清空队列，可以立即结束动画；gotoEnd，让当前正在执行的动画立即完成，并且重设 show 和 hide 的原始样式，调用回调函数等
delay(duration, [queueName])	$('#foo').slideUp(300). delay(800).fadeIn(400);	设置一个延时来推迟执行队列中之后的项目。用于将队列中的函数延时执行。它既可以推迟动画队列的执行，也可以用于自定义队列。duration，延时时间，单位为毫秒；queueName，队列名词，默认是 Fx，动画队列

除了上述这些函数外，jQuery 中与动画相关的还有一个设置选项 jQuery.fx.off，它关闭页面上所有的动画。把这个属性设置为 true 可以立即关闭所有动画（所有效果会立即执行完毕）。有些情况下可能需要这样，例如，

- 你在配置比较低的电脑上使用 jQuery。
- 你的一些用户由于动画效果而遇到了可访问性问题。

当把这个属性设成 false 之后，可以重新开启所有动画。

13.1.2　jQuery 动画简单示例

下面用一个自定义动画的简单示例来看一下 jQuery 制作动画的过程。在这个示例中我们所要达成的效果是：在页面上垂直摆放几个超链接，当将鼠标悬停在其中一个超链接上时，这个超链接动态向右缩进，当鼠标移开超链接时它恢复到原来的位置。实现原理就是利用了前面总结的自定义动画函数 animate()，并在这个函数中修改超链接的 padding-left 属性，而且给定了动画持续时间，效果如图 13.1 所示。

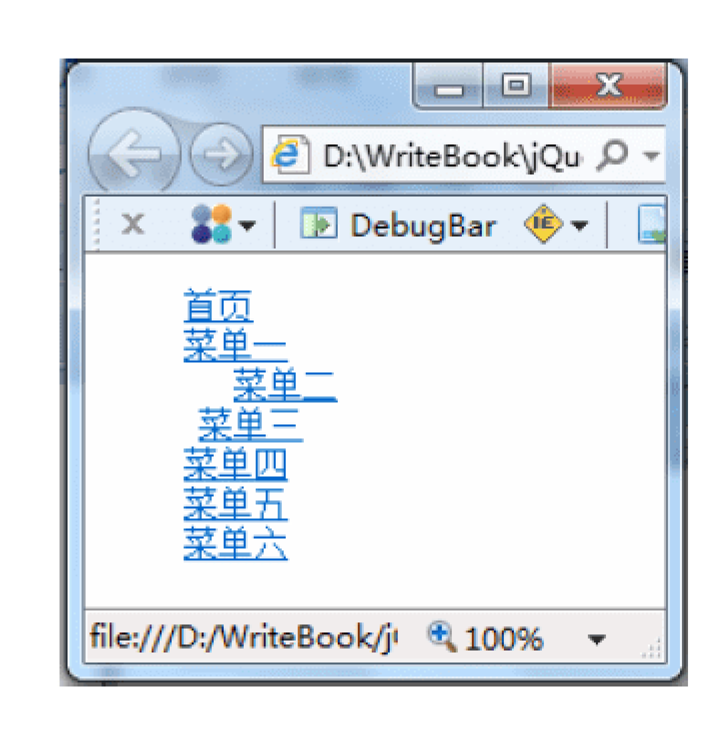


图 13.1　jQuery 简单动画效果一

为了实现这个效果，我们在 HTML 静态页面部分创建列表嵌套超链接：

```
1  <ul><li><a href="#" class="link">首页</a></li>
2    <li><a href="#" class="link">菜单一</a></li>
3     <li><a href="#" class="link">菜单二</a></li>
4     <li><a href="#" class="link">菜单三</a></li>
5     <li><a href="#" class="link">菜单四</a></li>
6     <li><a href="#" class="link">菜单五</a></li>
7     <li><a href="#" class="link">菜单六</a></li>
8   </ul>
```

然后，引入 jQuery 库文件：

```
<script type="text/javascript" src="jslib/jquery-1.6.js"></script>
```

最后，加上 JavaScript 功能代码：

```
1 <script type="text/javascript">
2   $(document).ready(function() {
3       $('a.link').hover(function() {
4           $(this).animate({ paddingLeft: '20px' },400);
                                             //调用自定义动画实现列表项右缩进
5       }, function() {
6           $(this).animate({ paddingLeft: 0 }, 400);
                                             //调用自定义动画实现列表项位置还原
7       });
8   });
9 </script>
```

上述代码第 3 行使用了模仿鼠标悬停与离开事件函数。第 4 行使用了自定义动画函数，

并设定超链接左缩进 20 像素的动画效果，整个动画持续时间为 400 毫秒，这个效果在鼠标悬停在超链接上时发生。第 6 行同样使用了自定义动画函数，将超链接位置恢复为原来的位置，动画时间持续为 400 毫秒，这个效果在鼠标离开超链接时发生。如果将鼠标依次滑过每个超链接，则动画效果会更清晰。效果如图 13.2 所示。



图 13.2　jQuery 简单动画效果二

13.2　jQuery UI 中实现的动画效果

在 jQuery UI 中提供了多种动画效果供我们参考。除了前面介绍的基本动画效果、滑动效果、淡入淡出效果外，还有百叶窗效果、跳跃效果、缩减效果、移动效果、分裂效果、折叠效果、高亮淡入淡出效果、脉冲闪烁效果、摆动效果等。本节逐一介绍它们的实现。

13.2.1　jQuery UI 动画特效使用

首先，参照 jQuery UI 中给出的示例来看一下如何使用动画特效。这个示例在 jQuery UI 的 demos 文件下的 show 子文件夹中。它使用了 jQuery UI 提供的各种特效显示一个消息层。

该示例的 HTML 源代码如下：

```
1 <div class="demo">
2 <div class="toggler">
3   <div id="effect" class="ui-widget-content ui-corner-all">
4       <h3 class="ui-widget-header ui-corner-all">Show</h3>
5       <p>
6           Etiam libero neque, luctus a, eleifend nec, semper at, lorem.
            Sed pede. Nulla lorem metus, adipiscing ut, luctus sed, hendrerit
            vitae, mi.
7       </p>
8   </div>
9 </div>
10 <select name="effects" id="effectTypes">
11  <option value="blind">Blind</option>
12  <option value="bounce">Bounce</option>
13  <option value="clip">Clip</option>
14  <option value="drop">Drop</option>
```

```
15  <option value="explode">Explode</option>
16  <option value="fold">Fold</option>
17  <option value="highlight">Highlight</option>
18  <option value="puff">Puff</option>
19  <option value="pulsate">Pulsate</option>
20  <option value="scale">Scale</option>
21  <option value="shake">Shake</option>
22  <option value="size">Size</option>
23  <option value="slide">Slide</option>
24 </select>
25  <a href="#" id="button" class="ui-state-default ui-corner-all">Run
    Effect</a>
26 </div><!-- End demo -->
27 <div class="demo-description">
28 <p>Click the button above to preview the effect.</p>
29 </div><!-- End demo-description -->
```

上述代码第 2～9 行是需要触发让其显示的信息层；第 10～24 层是选择各种效果的下拉列表框部分。

要实现这些动画效果，需要引入相应的 jQuery 库文件：

```
1   <script src="../../jquery-1.6.2.js"></script>
2   <script src="../../ui/jquery.effects.core.js"></script>
3   <script src="../../ui/jquery.effects.blind.js"></script>
4   <script src="../../ui/jquery.effects.bounce.js"></script>
5   <script src="../../ui/jquery.effects.clip.js"></script>
6   <script src="../../ui/jquery.effects.drop.js"></script>
7   <script src="../../ui/jquery.effects.explode.js"></script>
8   <script src="../../ui/jquery.effects.fold.js"></script>
9   <script src="../../ui/jquery.effects.highlight.js"></script>
10  <script src="../../ui/jquery.effects.pulsate.js"></script>
11  <script src="../../ui/jquery.effects.scale.js"></script>
12  <script src="../../ui/jquery.effects.shake.js"></script>
13  <script src="../../ui/jquery.effects.slide.js"></script>
```

上述各个 JS 文件就是 jQuery UI 提供动画特效的插件文件，每个插件文件名中都有 effects 这个单词，表示效果。

在 JavaScript 代码中按照下拉列表框中选择的效果运行插件：

```
1   <script>
2   $(function() {
3       //执行当前选择动画特效
4       function runEffect() {
5           //获取特效类型
6           var selectedEffect = $( "#effectTypes" ).val();
7           //很多特效不需要配置参数
8           var options = {};
9           //特效类型的相关参数设定
10          if ( selectedEffect === "scale" ) {
11              options = { percent: 100 };
12          } else if ( selectedEffect === "size" ) {
```

```
13              options = { to: { width: 280, height: 185 } };
14          }
15          //运行特效
16          $( "#effect" ).show( selectedEffect, options, 500, callback );
17      };
18      //回调函数使用了一个隐藏的消息层
19      function callback() {
20          setTimeout(function() {
21              $( "#effect:visible" ).removeAttr( "style" ).fadeOut();
22          }, 1000 );
23      };
24      //设置特效效果
25      $( "#button" ).click(function() {
26          runEffect();
27          return false;
28      });
29      $( "#effect" ).hide();
30  });
31  </script>
```

上述代码第 6 行获取下拉列表框的选择特效名称。第 8 行定义特效执行选项。第 10～14 行判断是使用原规模还是调整尺寸特效，如果使用原规模，则将缩放比例设成百分之百，如果是调整尺寸，则设定消息层宽和高。第 16 行利用 jQuery 的 show()函数调用特效，第 1 个参数是特效名称，第 2 个参数是特效选项，第 3 个参数是特效持续时间，第 4 个参数表示回调函数。第 19～23 行表示当特效把消息层显示出来后间隔一秒钟隐藏消息层。页面的初始加载效果如图 13.3 所示。

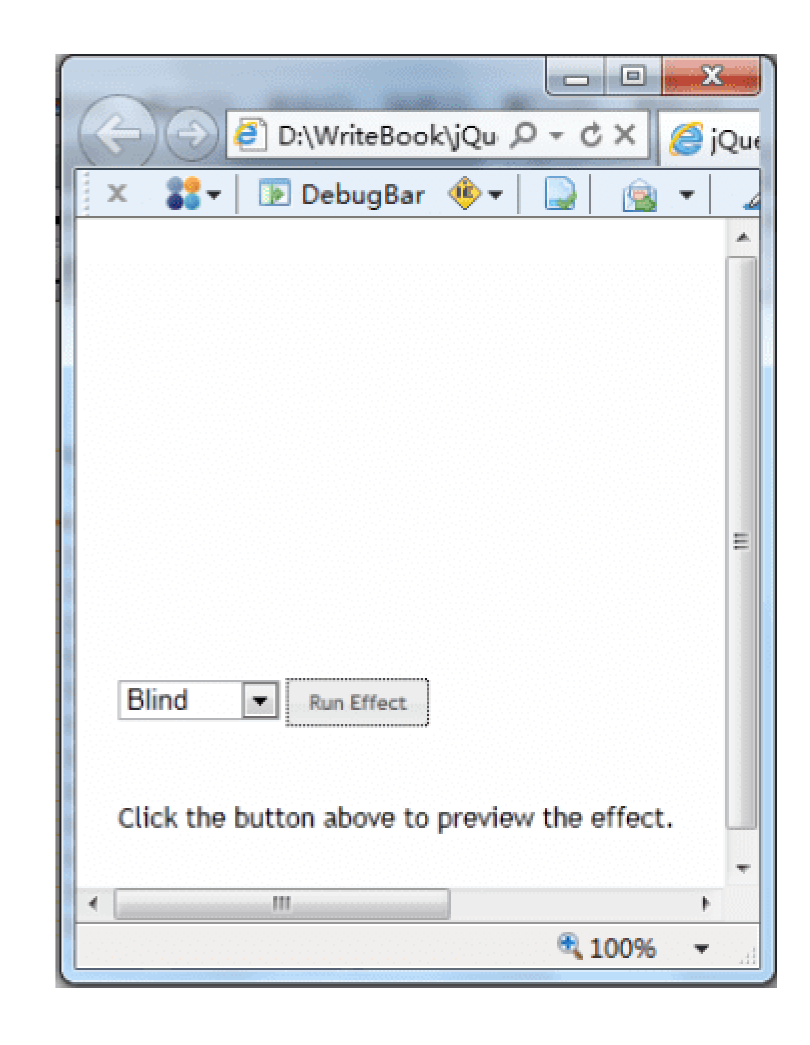


图 13.3　jQuery UI 特效初始页面

13.2.2　百叶窗效果

百叶窗效果使用了 jQuery UI 的 jquery.effects.core.js 和 jquery.effects.blind.js 文件。第一个文件是动画特效的核心文件，其中提供了一些实现特效的基本和常用操作函数。第二个文件是实现百叶窗效果的插件文件。它的设计思想是根据百叶窗打开方向设定动画的操作高度或者宽度，然后利用自定义动画实现动画效果。在这里使用了 jQuery 的 animate()自定义动画函数来完成动画效果。css()样式设定函数修改消息部分的样式，主要是宽和高两个参数。hide()为隐藏函数，将消息部分隐藏起来；height()为获取元素高度函数。show()为显示元素函数。width()为获取元素宽度函数。主要设计思路是根据动画显示方向（纵向或横向），通过自定义动画的特效将消息部分的高或者宽调整为原始大小。下面看一下百叶窗效果插件的功能代码：

```
1 (function( $, undefined ) {
2 $.effects.blind = function(o) {
3   return this.queue(function() {
4       //获取特效作用对象
```

```
5         var el = $(this), props = ['position','top','bottom','left',
          'right'];
6         //设置运行参数
7         var mode = $.effects.setMode(el, o.options.mode || 'hide');
          //设置对象的显示模式
8         var direction = o.options.direction || 'vertical';
          //设置动画的作用方向
9         //调整对象状态
10        $.effects.save(el, props); el.show();  //存储对象，并将对象显示出来
11        var wrapper = $.effects.createWrapper(el).css({overflow:'hidden'});
                                             //为对象创建一个外部封装
12        var ref = (direction == 'vertical') ? 'height' : 'width';
                                             //设定动画变化参考方向
          //设定包含封装元素的变化距离
13        var distance = (direction == 'vertical') ? wrapper.height() :
          wrapper.width();
14        if(mode == 'show') wrapper.css(ref, 0);//设定封装元素的初始状态
15        //使用自定义动画
16        var animation = {};
17        animation[ref] = mode == 'show' ? distance : 0;    //设定动画参数
18        //对封装元素使用自定义动画
19        wrapper.animate(animation, o.duration, o.options.easing, function() {
20            if(mode == 'hide') el.hide();       //隐藏元素
21            $.effects.restore(el, props); $.effects.removeWrapper(el);
                                        //恢复元素的初始状态，并移除封装元素
22            if(o.callback) o.callback.apply(el[0], arguments);
23            el.dequeue();
24        });
25    });
26 };
27 })(jQuery);
```

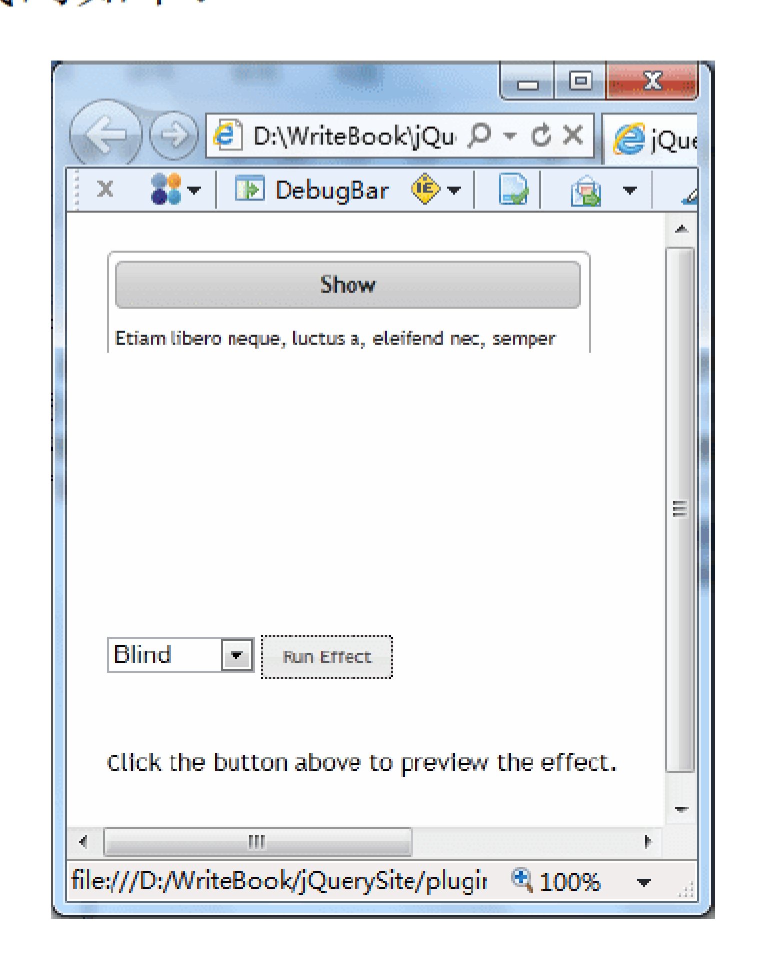
对于插件的编写要在第 15 章才会介绍，在这里只讨论它的实现过程。上述代码第 3 行返回当前调用插件的一个动画队列，并在这个队列中定义了回调函数实现百叶窗效果。第 5 行创建元素，接收调用插件的对象，并定义位置选项。第 7 行设置选项中的显示模式为隐藏。第 8 行设置选项中的动画方向为垂直。第 10 行保存元素对象的位置，并显示元素对象。第 11 行调用了 jquery.effects.core.js 文件中创建的包装元素函数<DIV>，并设定当元素内容溢出时隐藏。

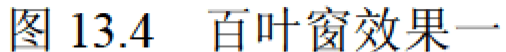
第 12 行根据动画方向选择修改宽或者高属性。第 13 行根据动画方向选择使用包装元素的宽或者高的属性值。第 14 行设置如果元素是显示的则隐藏。第 16 行定义动画参数数组。第 17 行根据显示状态设置设定动画参数数组的值。第 19 行调用包装元素的自定义动画函数。第 20 行判断显示模式，如果隐藏则调用元素对象的隐藏方法。第 21 恢复先前存储的元素属性。第 22 行调用回调函数。第 23 行将动画从队列删除。为了能够看清效果，我们可以将动画时间延长。效果如图 13.4 和图 13.5 所示。

13.2.3　跳跃效果

跳跃效果需要应用到动画特效的核心文件和跳跃效果的插件文件 jquery.effects.bounce.js。

它的设计思想是通过设定动画跳跃的行为参数、跳跃次数、跳跃高度、跳跃用时、循环调用自定义动画实现跳跃过程。所谓跳跃，实际上是通过变换设定元素的顶端坐标值来完成的。这里主要使用了 jQuery 的 animate()自定义动画函数、css()样式设定函数、show()显示元素函数。主要设计思路是通过不断改变消息部分的顶端坐标位置来实现跳跃动画效果。插件源代码如下：




图 13.4　百叶窗效果一

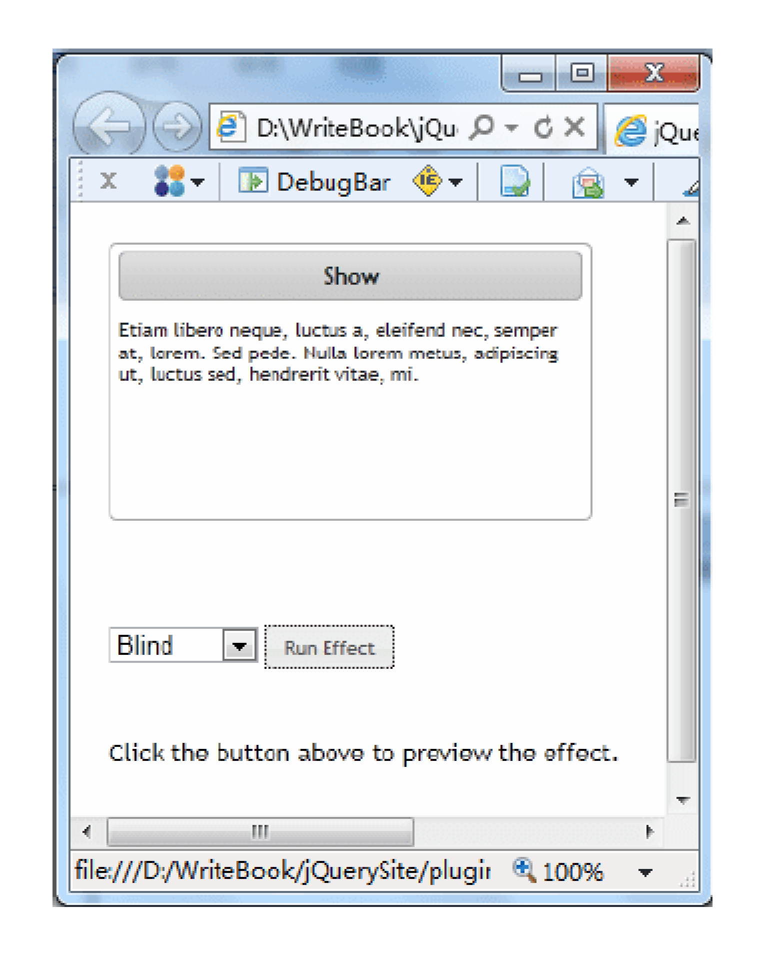


图 13.5　百叶窗效果二

```
1 (function( $, undefined ) {
2 $.effects.bounce = function(o) {
3   return this.queue(function() {
4       //获取特效作用对象
5       var el = $(this), props = ['position','top','bottom','left',
        'right'];
6       //设置动画运行参数
7       var mode = $.effects.setMode(el, o.options.mode || 'effect');
                                                        //设置对象的显示模式
8       var direction = o.options.direction || 'up';    //设置动画方向
9       var distance = o.options.distance || 20;        //默认动画运行距离
10      var times = o.options.times || 5;               //设置动画持续次数
11      var speed = o.duration || 250;                  //设置每次跳动速度
12      if (/show|hide/.test(mode)) props.push('opacity');//设置透明参数
13      //调整对象
14      $.effects.save(el, props); el.show();           //存储对象，并显示对象
15      $.effects.createWrapper(el);                    //创建对象的封装元素
16      var ref = (direction == 'up' || direction == 'down') ? 'top' : 'left';
                                                        //设定动画运行参考坐标
17      var motion = (direction == 'up' || direction == 'left') ? 'pos' : 'neg';
                                                        //设定动画动作参数
18      var distance = o.options.distance || (ref == 'top' ? el.outerHeight
        ({margin:true}) / 3 : el.outerWidth({margin:true}) / 3);
                                                  //设定动画初次实际运行距离
19      if (mode == 'show') el.css('opacity', 0).css(ref, motion == 'pos' ?
        -distance : distance);                    //设定样式
20      if (mode == 'hide') distance = distance / (times * 2);
                                                  //如果对象隐藏则动画距离缩减
21      if (mode != 'hide') times--;              //动画执行次数递减
```

```
22        // 使用自定义动画
23        if (mode == 'show') {                        //第一次显示跳跃动画
24            var animation = {opacity: 1};            //设置透明度
25            animation[ref] = (motion == 'pos' ? '+=' : '-=') + distance;
                                                       //设定初次向上跳跃
26            el.animate(animation, speed / 2, o.options.easing);//执行动画
27            distance = distance / 2;                 //跳动距离减半
28            times--;                                 //跳动次数自减
29        };
30        for (var i = 0; i < times; i++) {            //循环跳动 3 次
31            var animation1 = {}, animation2 = {};
32            animation1[ref] = (motion == 'pos' ? '-=' : '+=') + distance;
                                                       //向下跳动
33            animation2[ref] = (motion == 'pos' ? '+=' : '-=') + distance;
                                                       //向上跳动
34            el.animate(animation1, speed / 2, o.options.easing).animate
              (animation2, speed / 2, o.options.easing);
35            distance = (mode == 'hide') ? distance * 2 : distance / 2;
                                                       //跳动距离减半
36        };
37        if (mode == 'hide') {
38            var animation = {opacity: 0};
39            animation[ref] = (motion == 'pos' ? '-=' : '+=')  + distance;
40            el.animate(animation, speed / 2, o.options.easing, function(){
41                el.hide(); // Hide
42               $.effects.restore(el, props); $.effects.removeWrapper(el);
                                                       //移除外部封装
43                if(o.callback) o.callback.apply(this, arguments);
                                                       //回调方法
44            });
45        } else {                                     //最后一次跳动
46            var animation1 = {}, animation2 = {};
47            animation1[ref] = (motion == 'pos' ? '-=' : '+=') + distance;
48            animation2[ref] = (motion == 'pos' ? '+=' : '-=') + distance;
49            el.animate(animation1, speed / 2, o.options.easing).animate
              (animation2, speed / 2, o.options.easing, function(){
50               $.effects.restore(el, props); $.effects.removeWrapper(el);
                                                       //恢复对象
51                if(o.callback) o.callback.apply(this, arguments);
52            });
53        };
54        el.queue('fx', function() { el.dequeue(); });
55        el.dequeue();
56    });
57  };
58  })(jQuery);
```

上述代码第 7 行设置显示模式。第 8～11 行设定跳跃行为参数、跳动方向、跳动距离、跳动次数、跳动间隔。第 12 行避免 IE 对于透明度的问题设置。第 13、14 行保存元素对象属性并显示。第 15 行创建包装标记。第 16 行根据跳跃方向选择坐标类型。第 17 行根据跳跃方向选择操作参数。第 18 行获取跳跃距离。第 19 行设置如果元素为显示状态则转换。

第 20 行设置如果元素为隐藏状态则距离递减。

第 21 行设置如果模式不是隐藏则跳跃次数减 1。第 23～29 行利用自定义动画行执行跳跃动画。第 30～36 行利用自定义动画上下跳跃循环。第 37～45 行利用自定义动画最后一次跳跃，并恢复元素属性，移除包装元素，调用回调函数。第 46～53 行利用自定义动画继续跳跃，并恢复元素属性，移除包装元素，调用回调函数。第 55、56 行从队列删除动画。效果如图 13.6 所示。

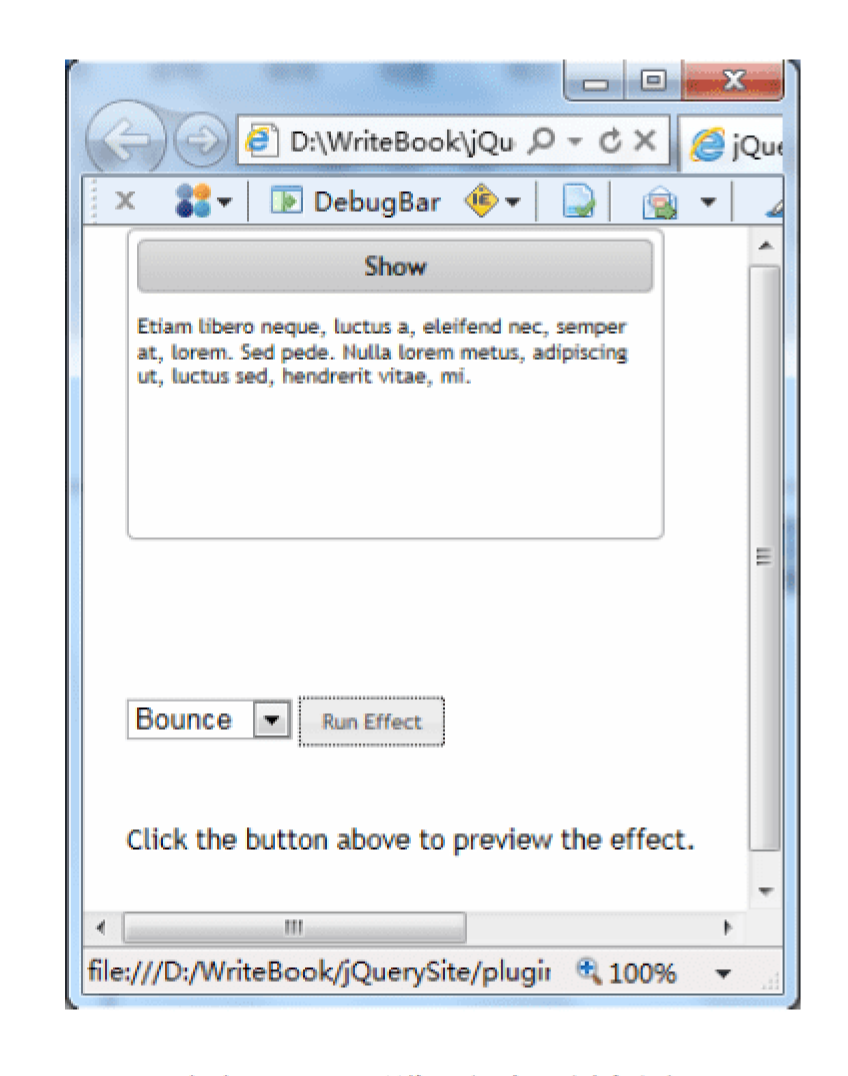


图 13.6　跳跃动画效果

13.2.4　缩减效果

缩减效果需要应用到动画特效的核心文件和缩减效果的插件文件 jquery.effects.clip.js。它的设计思想是设定元素起始出现位置，即元素高度的一半，然后元素的上下两部分依次向两边展开。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.clip = function(o) {
3   return this.queue(function() {
4       //获取动画作用对象
5       var el = $(this), props = ['position','top','bottom','left','right',
        'height','width'];
6       //设置对象运行参数
7       var mode = $.effects.setMode(el, o.options.mode || 'hide');
                                                          //设置显示方式
8       var direction = o.options.direction || 'vertical'; //默认动画方向
9       // 调整对象状态
10      $.effects.save(el, props); el.show();             //存储对象并显示
11      var wrapper = $.effects.createWrapper(el).css({overflow:'hidden'});
                                                          //创建封装元素
12      var animate = el[0].tagName == 'IMG' ? wrapper : el;
                                                      //获取动画作用对象
13      var ref = {                                   //动画运行参数
14          size: (direction == 'vertical') ? 'height' : 'width',
15          position: (direction == 'vertical') ? 'top' : 'left'
16      };
17      var distance = (direction == 'vertical') ? animate.height() : animate.
        width();                                      //获取动画运行距离
18      if(mode == 'show') { animate.css(ref.size, 0); animate.css(ref.
        position,distance / 2); }                     //初始动画对象
19      //使用自定义动画对象
20      var animation = {};
21      animation[ref.size] = mode == 'show' ? distance : 0;
                                              //设定完整显示对象高度
22      animation[ref.position] = mode == 'show' ? 0 : distance / 2;
                                              //设定开始动画位置，为高度一半
23      //使用自定义动画
```

```
24      animate.animate(animation, { queue: false, duration: o.duration,
        easing: o.options.easing, complete: function() {
25          if(mode == 'hide') el.hide();       //隐藏对象
26          $.effects.restore(el, props); $.effects.removeWrapper(el);
                                                //恢复对象，并移除封装
27          if(o.callback) o.callback.apply(el[0], arguments);
28          el.dequeue();
29      }});
30  });
31 };
32 })(jQuery);
```

上述代码同百叶窗效果类似，也是使用包装元素<DIV>实现消息层的大小动画。但是，这个插件的显示是从消息层垂直方向的中间位置向上下扩展，效果如图 13.7 所示。

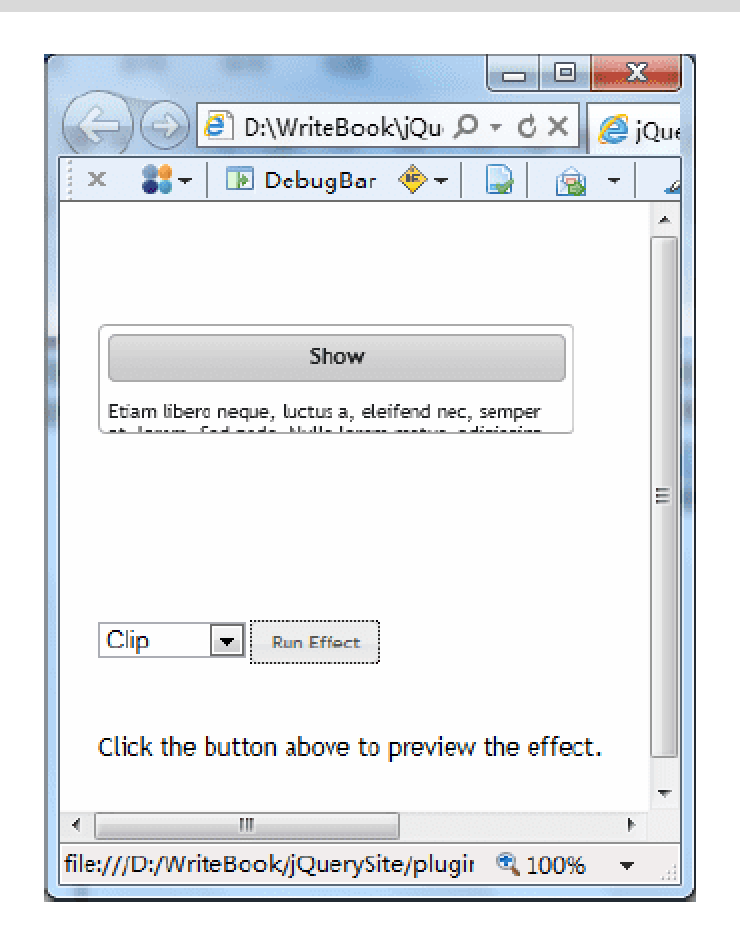


图 13.7　缩减动画效果

13.2.5　移动效果

移动效果需要应用到动画特效的核心文件和移动效果的插件文件 jquery.effects.drop.js。它的设计思想类似于跳跃效果，只是在移动过程中通过一次自定义动画将元素起始左端位置加上一定的像素值，使元素向右移动。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.drop = function(o) {
3   return this.queue(function() {
4       //获取动画作用对象
5       var el = $(this), props = ['position','top','bottom','left','right',
        'opacity'];
6       //设定动画参数
7       var mode = $.effects.setMode(el, o.options.mode || 'hide');
                                                        //设定对象显示模式
8       var direction = o.options.direction || 'left'; //默认动画移动方向
9       //调整对象
10      $.effects.save(el, props); el.show();          //存储原对象并显示
11      $.effects.createWrapper(el);                   //创建对象的封装
12      var ref = (direction == 'up' || direction == 'down') ? 'top' : 'left';
                                                //创建动画运行参考坐标方向
13      var motion = (direction == 'up' || direction == 'left') ? 'pos' : 'neg';
                                                //创建动画动作参数
14      var distance = o.options.distance || (ref == 'top' ? el.outerHeight
        ({margin:true}) / 2 : el.outerWidth({margin:true}) / 2);
                                                //设定动画实际运动距离
        //转换对象初始状态
15      if (mode == 'show') el.css('opacity', 0).css(ref, motion == 'pos' ?
        -distance : distance);
16      // 使用自定义动画
```

```
17      var animation = {opacity: mode == 'show' ? 1 : 0};
                                        //设定动画中需要改变的透明度
18      animation[ref] = (mode == 'show' ? (motion == 'pos' ? '+=' : '-=') :
        (motion == 'pos' ? '-=' : '+=')) + distance;
                                        //设定动画运动最终目标参数值
19      //运行动画
20      el.animate(animation, { queue: false, duration: o.duration, easing:
        o.options.easing, complete: function()
21 {
22          if(mode == 'hide') el.hide();       //隐藏对象
23          $.effects.restore(el, props); $.effects.removeWrapper(el);
                                        //恢复对象，移除封装
24          if(o.callback) o.callback.apply(this, arguments);
25          el.dequeue();
26      }});
27  });
28 };
29 })(jQuery);
```

上述代码第 8 行设定动画移动方向向左。第 13 行设定如果动画移动方向向左，设定操作参数。第 14 行设定动画移动距离。第 15 行转换移动位移为负值，实际向右移动。第 17 行设定透明度。第 18 行设定自定义动画参数。第 20～26 行执行自定义动画并恢复元素属性，调用回调函数。效果如图 13.8 所示。

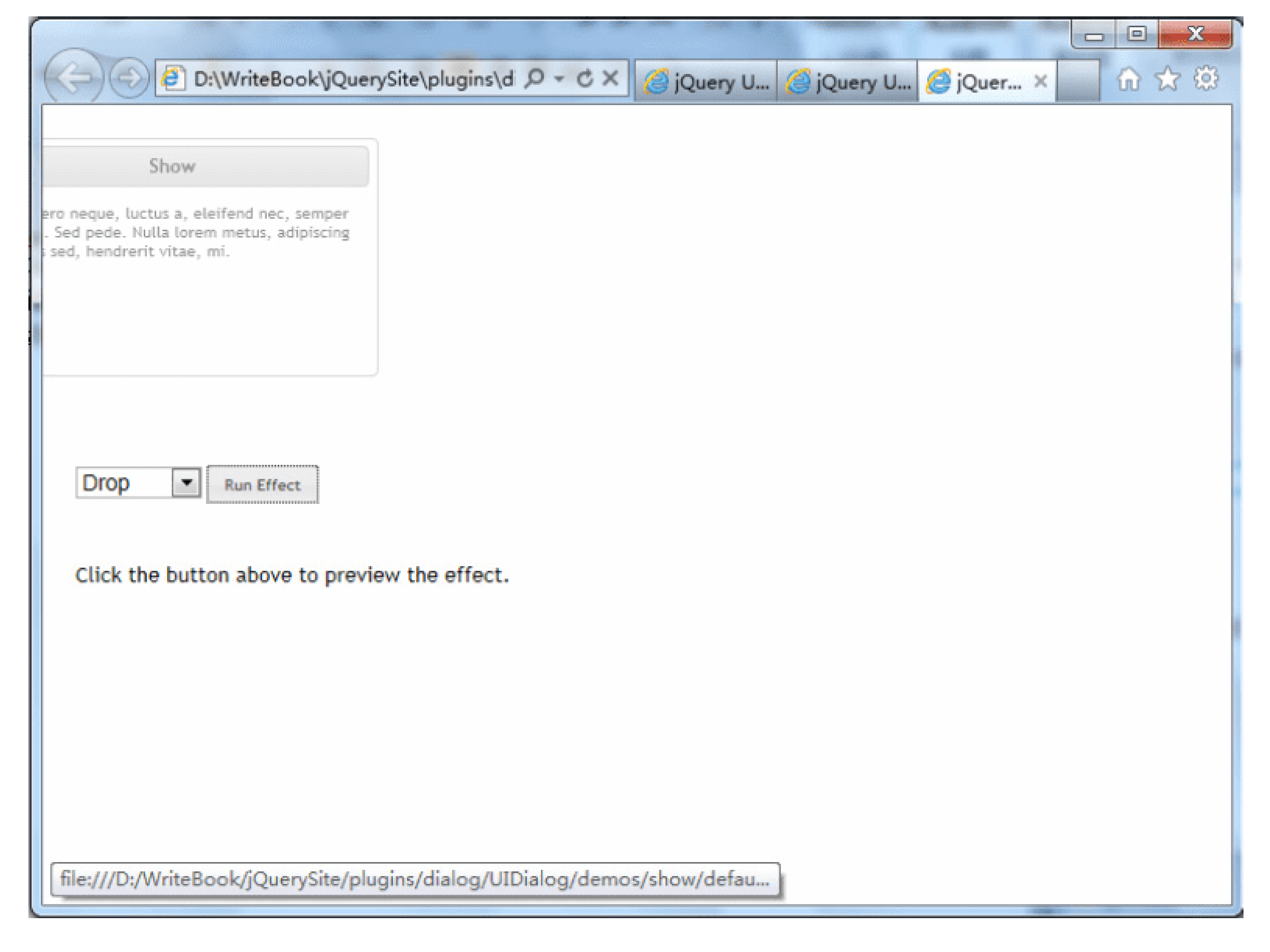


图 13.8　移动动画效果

13.2.6　分裂效果

分裂效果需要应用到动画特效的核心文件和分裂效果的插件文件 jquery.effects.explode.js。它的设计思想是复制多个元素，元素个数由动画排列的行数和列数决定，将复制出的多个元素通过自定义动画变换坐标位置由显示区域远端向中间移动完成效果。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.explode = function(o) {
3   return this.queue(function() {
4   var rows = o.options.pieces ? Math.round(Math.sqrt(o.options.pieces)) : 3;
                                                    //获取动画中行的个数
5   var cells = o.options.pieces ? Math.round(Math.sqrt(o.options.pieces)) : 3;
                                                    //获取动画中列的个数
    //设定动画显示参数
6   o.options.mode = o.options.mode == 'toggle' ? ($(this).is(':visible') ?
    'hide' : 'show') : o.options.mode;
7   var el = $(this).show().css('visibility', 'hidden');
                                                    //获取动画作用对象并显示
8   var offset = el.offset();                       //取得对象相对于浏览器的坐标位置
9   //设定左上角坐标位置
10  offset.top -= parseInt(el.css("marginTop"),10) || 0;
11  offset.left -= parseInt(el.css("marginLeft"),10) || 0;
12  var width = el.outerWidth(true);                //元素宽度
13  var height = el.outerHeight(true);              //元素高度
14  for(var i=0;i<rows;i++) {                       //按照行数循环
15      for(var j=0;j<cells;j++) {                  //按照列数循环
16          el
17              .clone()                            //克隆对象
18              .appendTo('body')                   //添加到页面中
19              .wrap('<div></div>')                //为每一个复制出来的元素用层封装
20              .css({
21                  position: 'absolute',
22                  visibility: 'visible',
23                  left: -j*(width/cells),
24                  top: -i*(height/rows)
25              })//设定每个复制元素的位置并根据行和列位置设定高和宽
26              .parent()                                   //获取封装层
27              .addClass('ui-effects-explode')             //在层上加入样式设定
28              .css({                                      //设定封装层的位置及尺寸
29                  position: 'absolute',
30                  overflow: 'hidden',
31                  width: width/cells,
32                  height: height/rows,
33                  left: offset.left + j*(width/cells) + (o.options.mode
                    == 'show' ? (j-Math.floor(cells/2))*(width/cells) : 0),
34                  top: offset.top + i*(height/rows) + (o.options.mode ==
                    'show' ? (i-Math.floor(rows/2))*(height/rows) : 0),
35                  opacity: o.options.mode == 'show' ? 0 : 1
36              }).animate({            //利用自定义动画变化封装层的位置
37                  left: offset.left + j*(width/cells) + (o.options.mode
                    == 'show' ? 0 : (j-Math.floor(cells/2))*(width/cells)),
38                  top: offset.top + i*(height/rows) + (o.options.mode ==
                    'show' ? 0 : (i-Math.floor(rows/2))*(height/rows)),
39                  opacity: o.options.mode == 'show' ? 1 : 0
40              }, o.duration || 500);
41      }
42  }
43  //当动画完成设定定时函数
```

```
44  setTimeout(function() {
    //设定对象是否显示
45      o.options.mode == 'show' ? el.css({ visibility: 'visible' }) : el.
        css({ visibility: 'visible' }).hide();
46              if(o.callback) o.callback.apply(el[0]);
47              el.dequeue();
48              $('div.ui-effects-explode').remove();  //移除封装层
49  }, o.duration || 500);
50  });
51};
52})(jQuery);
```

上述代码第 4、5 行设定了分裂显示时的行和列的数目。第 6 行设定显示模式。第 7 行获取隐藏元素对象。第 8 行获取元素相对于浏览器窗口的坐标位置。第 10、11 行将元素的左上角坐标值分别减去边界空白宽度。第 12、13 行获取元素的外部宽和外部高，包括边框和边界空白。第 14～42 行根据行和列的个数循环复制多个消息部分，并设定每个复制出来的消息层的起始位置，然后使用自定义动画函数，将每个消息层显示在指定位置，拼凑成一个完整的消息层。第 44～49 行设定一个超时函数，实现消息层的显示状态转换，并调用回调函数，删除队列，移除 CSS 样式设定。效果如图 13.9 所示。

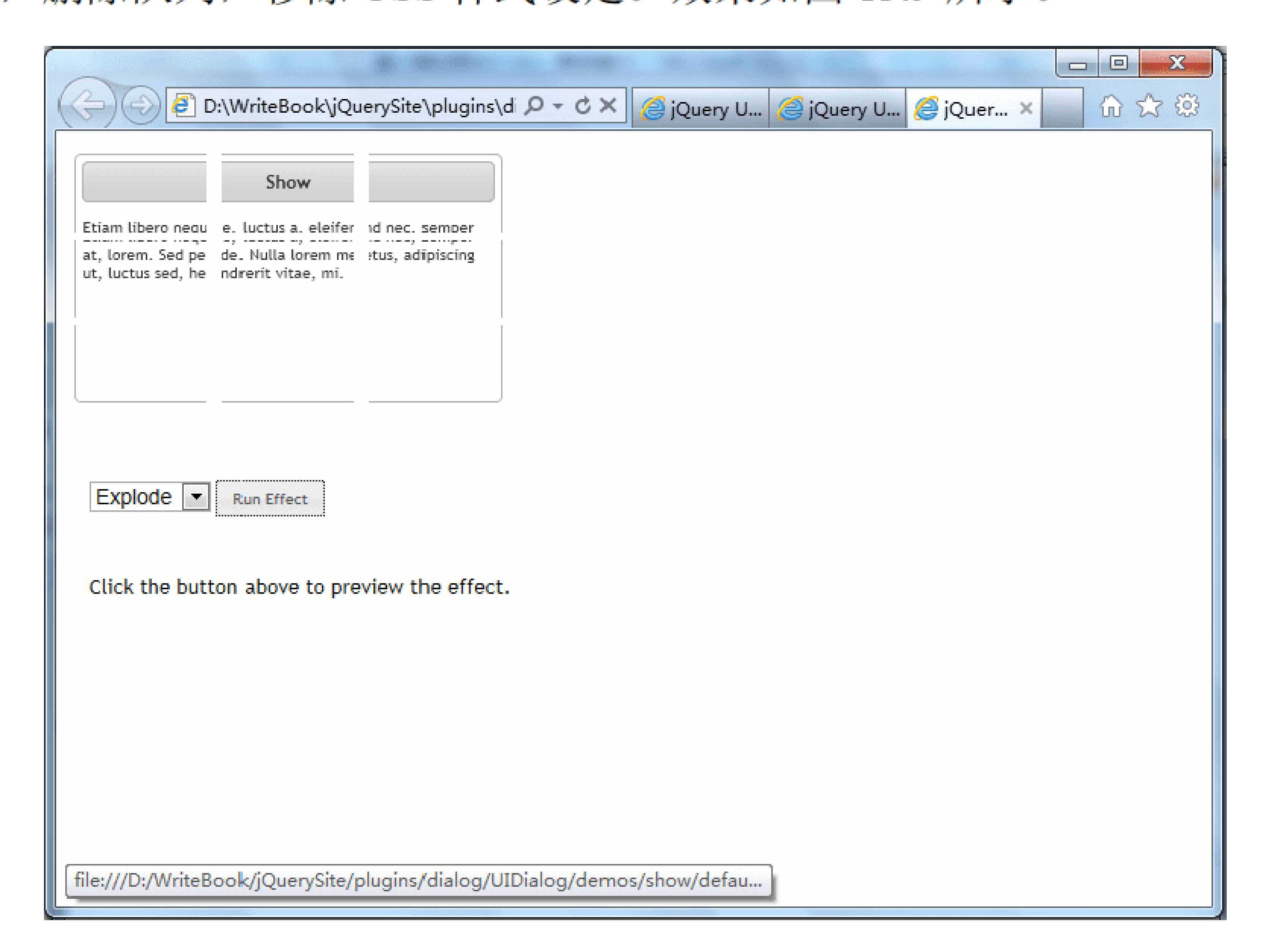


图 13.9　移动动画效果

13.2.7　折叠效果

折叠效果需要应用到动画特效的核心文件和折叠效果的插件文件 jquery.effects.fold.js。它的设计思想是按照一定的折叠方向和折叠比例，将元素用自定义动画多次修改坐标位置实现。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.fold = function(o) {
```

```
3   return this.queue(function() {
4       // 获取动画作用的对象
5       var el = $(this), props = ['position','top','bottom','left',
        'right'];
6       // 设定动画参数
7       var mode = $.effects.setMode(el, o.options.mode || 'hide');
                                                //设定对象显示模式
8       var size = o.options.size || 15;        //设定默认折叠大小
9       var horizFirst = !(!o.options.horizFirst); //是否是起始水平折叠
10      var duration = o.duration ? o.duration / 2 : $.fx.speeds. default / 2;
                                                //动画持续时间
11      // 调整对象
12      $.effects.save(el, props); el.show();   //存储对象并显示
13      var wrapper = $.effects.createWrapper(el).css({overflow:'hidden'});
                                                //创建封装元素
14      var widthFirst = ((mode == 'show') != horizFirst);
                                                //设定是否是首次修改宽度
15      var ref = widthFirst ? ['width', 'height'] : ['height', 'width'];
                                                //设定动画操作相关参数
                                    //根据封装元素的宽或者高设定动画距离
16      var distance = widthFirst ? [wrapper.width(), wrapper.height()] :
        [wrapper.height(), wrapper.width()];
17      var percent = /([0-9]+)%/.exec(size);   //计算百分比
18      if(percent) size = parseInt(percent[1],10) / 100 * distance[mode ==
        'hide' ? 0 : 1];                        //设定折叠距离
19     if(mode == 'show') wrapper.css(horizFirst ? {height: 0, width: size} :
        {height: size, width: 0});              //初始状态
20      // 使用动画
21      var animation1 = {}, animation2 = {};
22      animation1[ref[0]] = mode == 'show' ? distance[0] : size;
                                                //横向展开参数
23      animation2[ref[1]] = mode == 'show' ? distance[1] : 0;
                                                //纵向展开参数
24      // 调用自定义动画
25      wrapper.animate(animation1, duration, o.options.easing)
26      .animate(animation2, duration, o.options.easing, function() {
27          if(mode == 'hide') el.hide();       //隐藏元素
28          $.effects.restore(el, props); $.effects.removeWrapper(el);
                                                //恢复对象并移除封装
29          if(o.callback) o.callback.apply(el[0], arguments);
30          el.dequeue();
31      });
32  });
33};
34})(jQuery);
```

上述代码第 8 行设定默认折叠效果大小。第 9 行确认首次折叠是水平方向折叠。第 10 行设定动画持续时间。第 14、15 行确认两次折叠的先后参数。第 16 行确认两次折叠的先后参数值。第 17 行确认折叠比例百分比。第 18 行确认折叠效果大小。第 19 行转换包装后的元素样式。第 25～31 行调用自定义动画实现折叠打开效果，并删除包装元素，调用回调函数，从队列中删除。效果如图 13.10 和图 13.11 所示。

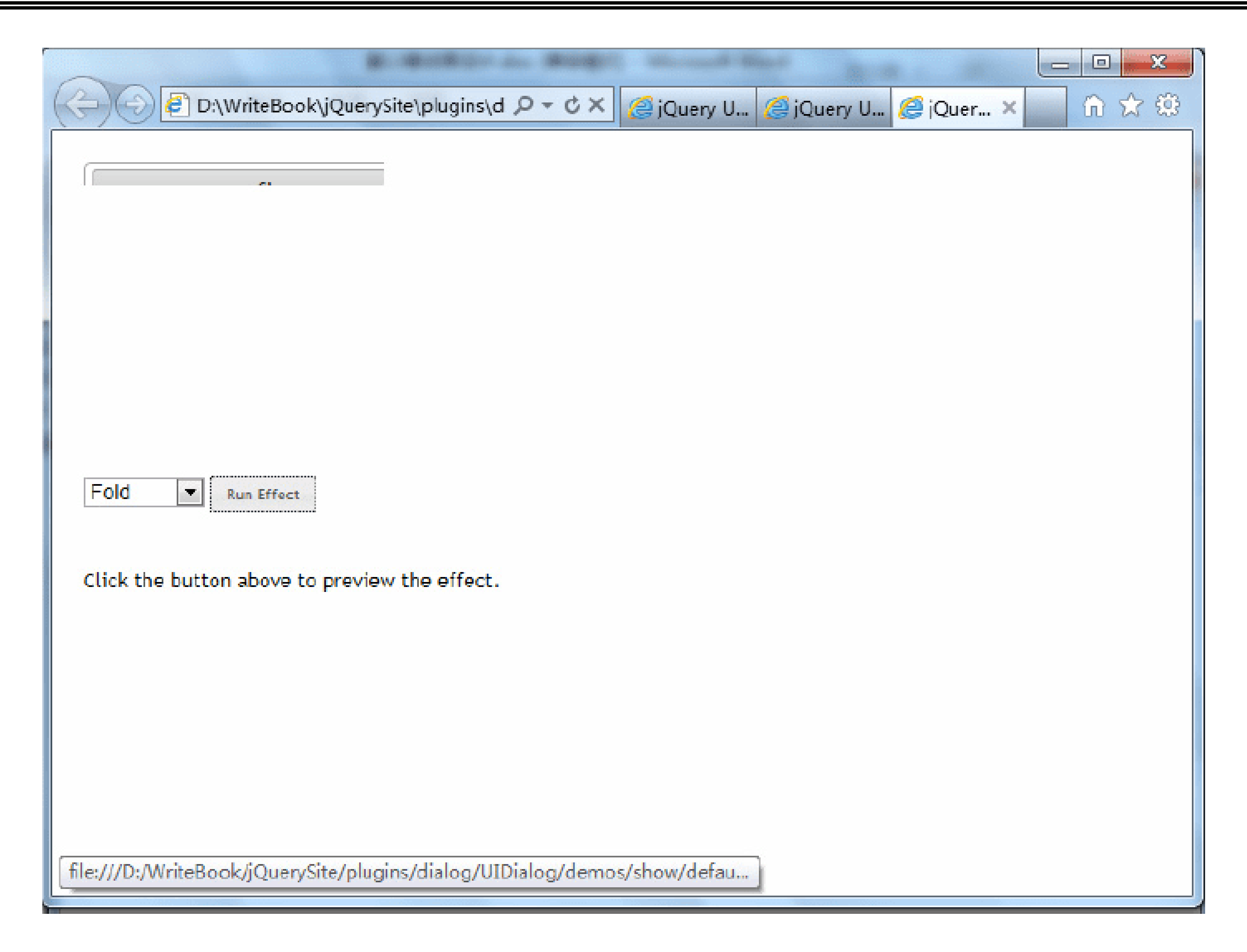


图 13.10　折叠动画效果横向打开

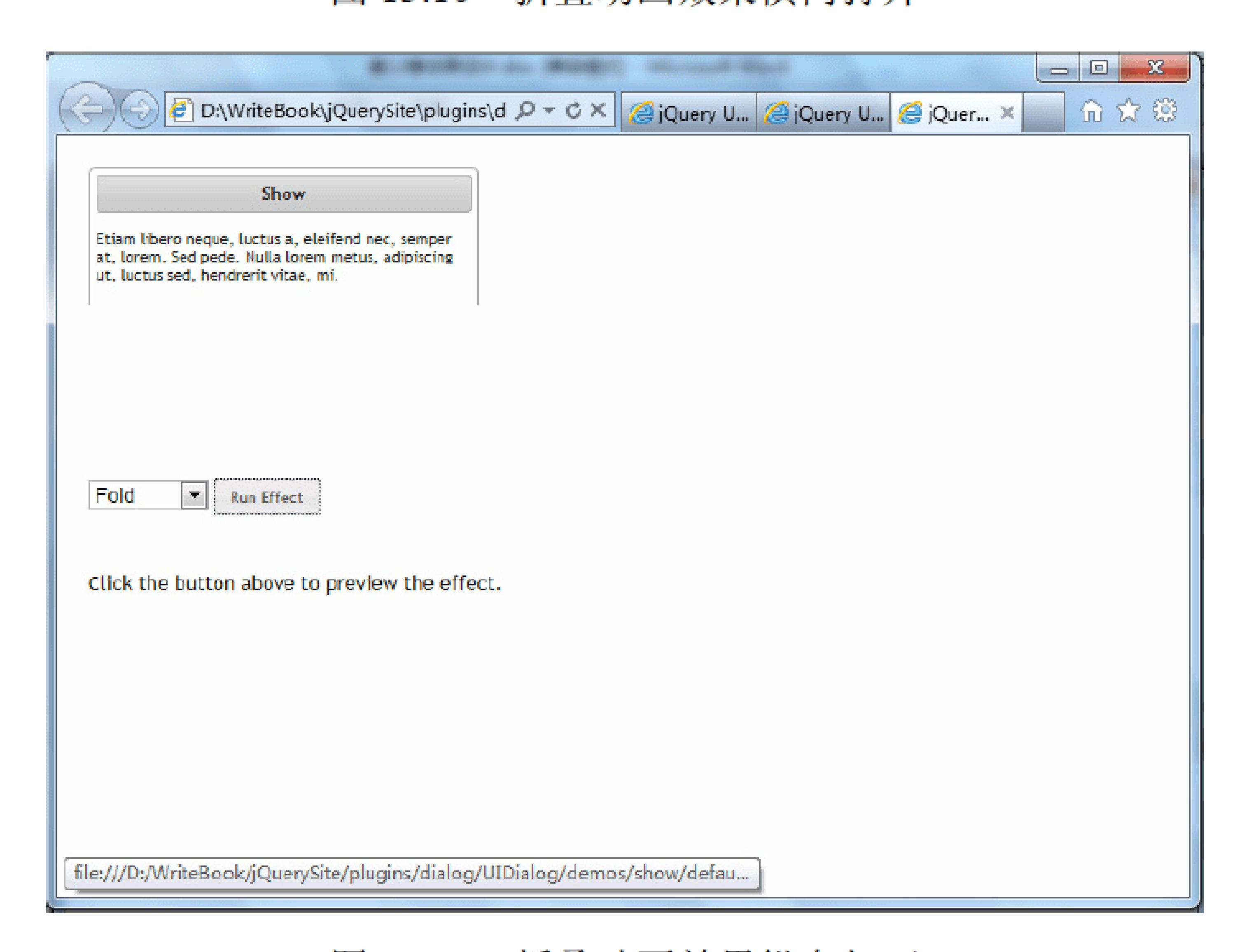


图 13.11　折叠动画效果纵向打开

13.2.8　高亮淡入淡出效果

高亮淡入淡出效果需要应用到动画特效的核心文件和高亮效果的插件文件 jquery.effects.highlight.js。它的设计思想比较简单，就是频繁更换背景色及透明度来实现效果。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.highlight = function(o) {
3   return this.queue(function() {
4       var elem = $(this),                             //获取动画作用对象
5           props = ['backgroundImage', 'backgroundColor', 'opacity'],
6           mode = $.effects.setMode(elem, o.options.mode || 'show'),
                                                        //设置对象显示模式
7           animation = {
```

```
8               backgroundColor: elem.css('backgroundColor')
9           };                                          //动画背景色参数
10      if (mode == 'hide') {                           //动画透明度参数
11          animation.opacity = 0;
12      }
13      $.effects.save(elem, props);                    //存储对象
14      elem
15          .show()                                     //显示对象
16          .css({
17              backgroundImage: 'none',
18              backgroundColor: o.options.color || '#ffff99'
19          })                                          //去除背景图片，设定背景色
20          .animate(animation, {                       //自定义动画效果
21              queue: false,
22              duration: o.duration,
23              easing: o.options.easing,
24              complete: function() {
25                  (mode == 'hide' && elem.hide());
26                  $.effects.restore(elem, props);
27                  (mode == 'show' && !$.support.opacity && this.style.
                    removeAttribute('filter'));
28                  (o.callback && o.callback.apply(this, arguments));
29                  elem.dequeue();
30              }
31          });
32  });
33};
34})(jQuery);
```

上述代码第 7～9 行设定动画需要的背景色参数。第 18 行设定背景色。第 20～31 行调用自定义动画，更换背景色，调用回调函数，删除队列。效果如图 13.12 所示。

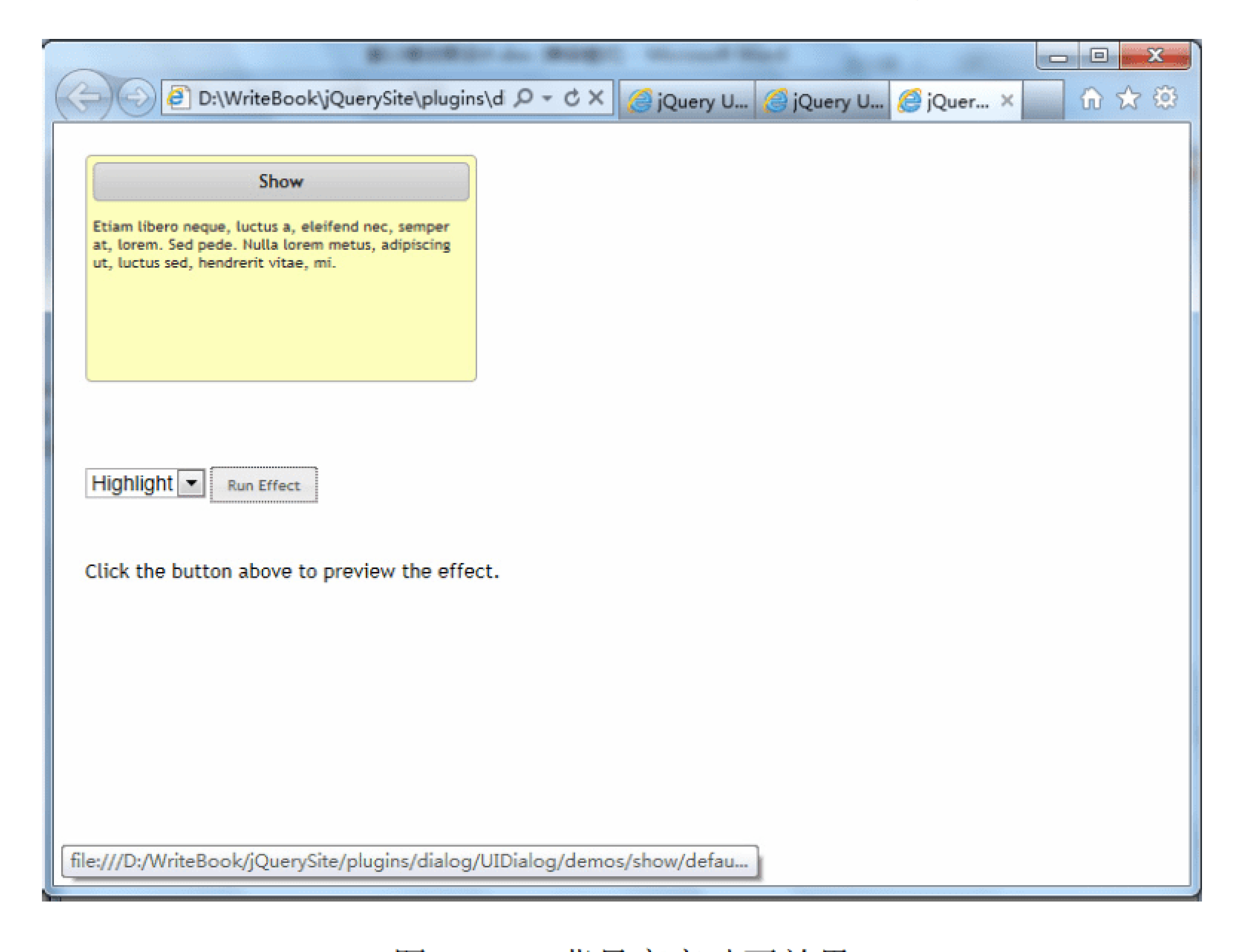


图 13.12　背景高亮动画效果

13.2.9　脉冲闪烁效果

脉冲闪烁效果需要应用到动画特效的核心文件和闪烁效果的插件文件 jquery.effects.pulsate.js。它的设计思想是利用自定义动画函数频繁变化元素透明度来完成效果。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.pulsate = function(o) {
3   return this.queue(function() {
4       var elem = $(this), //获取动画作用对象
5           mode = $.effects.setMode(elem, o.options.mode || 'show');
                                                    //设置对象显示模式
6           times = ((o.options.times || 5) * 2) - 1;  //设定动画次数
7           duration = o.duration ? o.duration / 2 : $.fx.speeds. default / 2,
                                                    //设定动画持续时间
8           isVisible = elem.is(':visible'),         //判断对象的可见状态
9           animateTo = 0;                           //显示状态切换标志
10      if (!isVisible) {
11          elem.css('opacity', 0).show();           //显示对象
12          animateTo = 1;                           //显示状态
13      }
14      if ((mode == 'hide' && isVisible) || (mode == 'show' && !isVisible)) {
15          times--;                                 //动画次数自减
16      }
17      for (var i = 0; i < times; i++) {
18          elem.animate({ opacity: animateTo }, duration, o.options.easing);
                                              //自定义动画调整对象透明度
19          animateTo = (animateTo + 1) % 2;   //更改显示状态标志
20      }
        //最后操作动画
21      elem.animate({ opacity: animateTo }, duration, o.options.easing,
        function() {
22          if (animateTo == 0) {
23              elem.hide();
24          }
25          (o.callback && o.callback.apply(this, arguments));
26      });
27      elem
28          .queue('fx', function() { elem.dequeue(); })
29          .dequeue();
30  });
31};
32})(jQuery);
```

上述代码第 6 行设定闪烁次数。第 7 行设定闪烁间隔。第 10～13 行判断元素是否可见，如果不可见则透明度为零，调用显示，设定透明度变量值为 1。第 14～16 行修改闪烁次数。第 17～20 行根据循环次数调用自定义动画函数交叉修改元素的可见性。第 21～26 行调用自定义动画函数设置透明度，并调用回调函数。效果如图 13.13 所示。

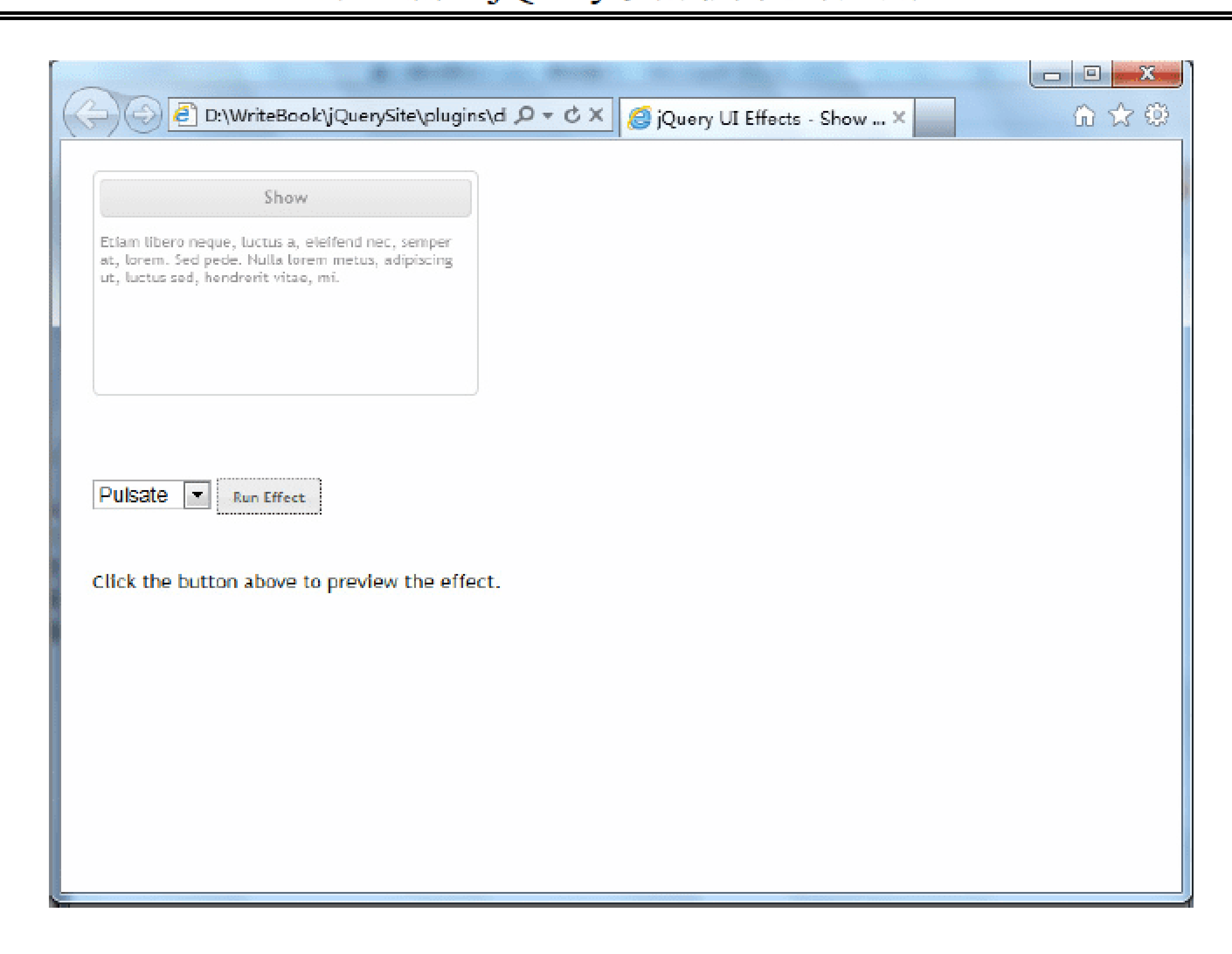


图 13.13　脉冲闪烁动画效果

13.2.10　摆动效果

摆动效果需要应用到动画特效的核心文件和摆动效果的插件文件 jquery.effects.shake.js。它的设计思想和跳跃效果基本相同，只是摆动方向为左右方向，并且在摆动过程中的第一次摆动、最后一次摆动及中间的摆动距离考虑的设定比跳跃效果复杂。插件源代码如下：

```
1 (function( $, undefined ) {
2 $.effects.shake = function(o) {
3   return this.queue(function() {
4       //获取动画作用对象
5       var el = $(this), props = ['position','top','bottom','left',
        'right'];
6       //设定动画运行参数
7       var mode = $.effects.setMode(el, o.options.mode || 'effect');
                                                    //设定效果模式
8       var direction = o.options.direction || 'left'; //设定默认移动方向
9       var distance = o.options.distance || 20;      //设定默认移动距离
10      var times = o.options.times || 3;             //设定动画次数
11      var speed = o.duration || o.options.duration || 140;
                                                    //设定默认动画速度
12      //调整对象
13      $.effects.save(el, props); el.show();         // 存储并显示对象
14      $.effects.createWrapper(el);                  // 创建封装对象
15      var ref = (direction == 'up' || direction == 'down') ? 'top' : 'left';
                                                    //创建动画作用方向参数
16      var motion = (direction == 'up' || direction == 'left') ? 'pos' : 'neg';
                                                    //设定运动方式
17      //使用自定动画
18      var animation = {}, animation1 = {}, animation2 = {};
```

```
19      animation[ref] = (motion == 'pos' ? '-=' : '+=')  + distance;
                                                    //设定最后一次左向摆动距离值
20      animation1[ref] = (motion == 'pos' ? '+=' : '-=')  + distance * 2;
                                                    //设定最后一次右向摆动距离值
21      animation2[ref] = (motion == 'pos' ? '-=' : '+=')  + distance * 2;
                                                    //设定最后一次左向摆动距离值
22      //首次调用自定义动画参数
23      el.animate(animation, speed, o.options.easing);
24      for (var i = 1; i < times; i++) {  //循环调用自定义动画，实现抖动效果
25          el.animate(animation1, speed, o.options.easing).animate
            (animation2, speed, o.options.easing);
26      };
27      el.animate(animation1, speed, o.options.easing).
28      animate(animation, speed / 2, o.options.easing, function(){
                                                //最后一次抖动效果
29          $.effects.restore(el, props); $.effects.removeWrapper(el);
                                                //恢复对象并移除封装
30          if(o.callback) o.callback.apply(this, arguments); // Callback
31      });
32      el.queue('fx', function() { el.dequeue(); });
33      el.dequeue();
34   });
35};
36})(jQuery);
```

上述代码第 8 行设定默认摆动方向。第 9 行设定摆动距离。第 10 行设定摆动次数。第 11 行设定动画持续时间。第 19～21 行设定三个动画摆动距离。第 23 行设定第一次摆动向右。第 24～26 行设定根据摆动次数多次摆动，但摆动距离比第一次多一倍。第 27～31 行设定最后一次摆动，并调用回调函数。效果如图 13.14 和图 13.15 所示。

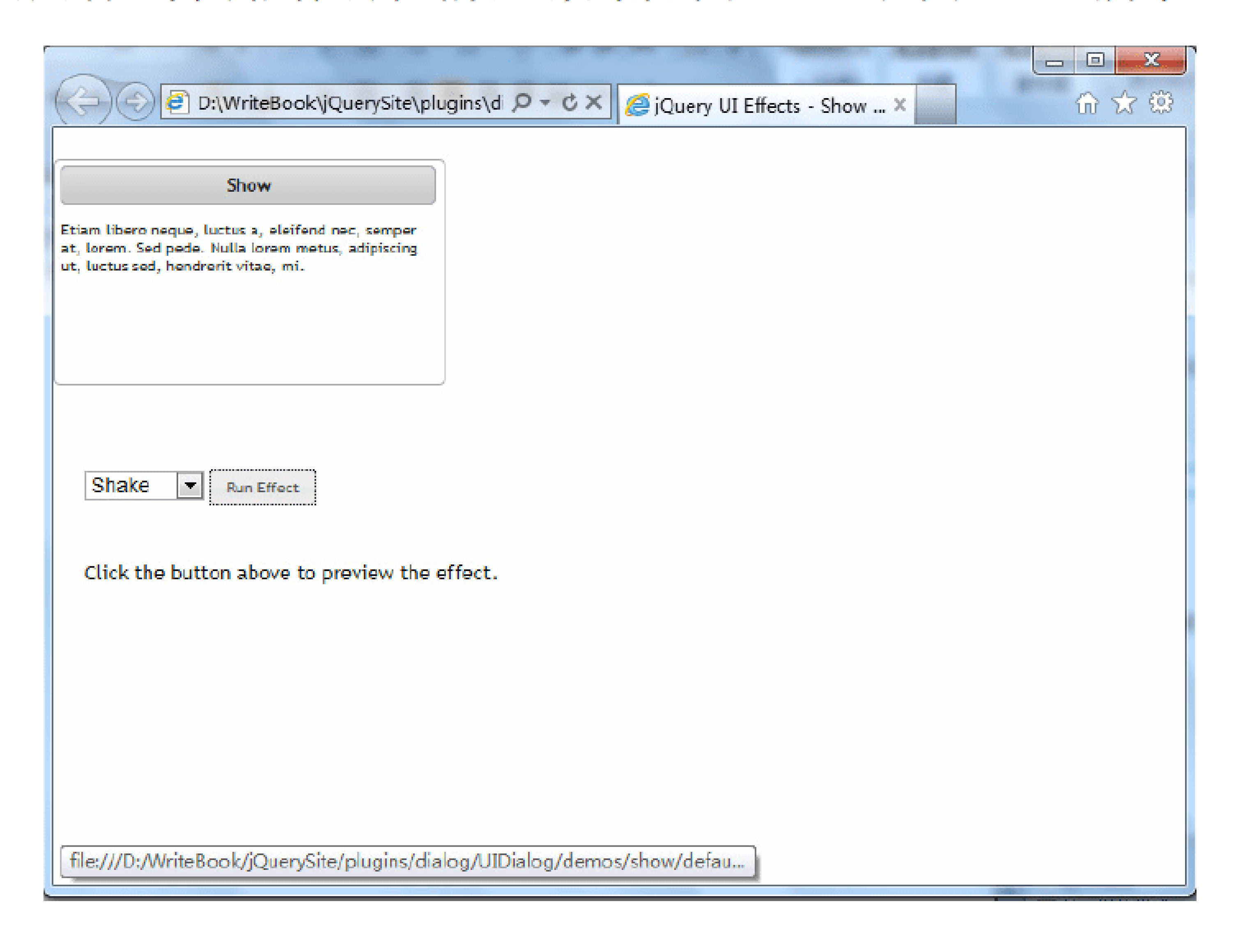


图 13.14　摆动动画效果一

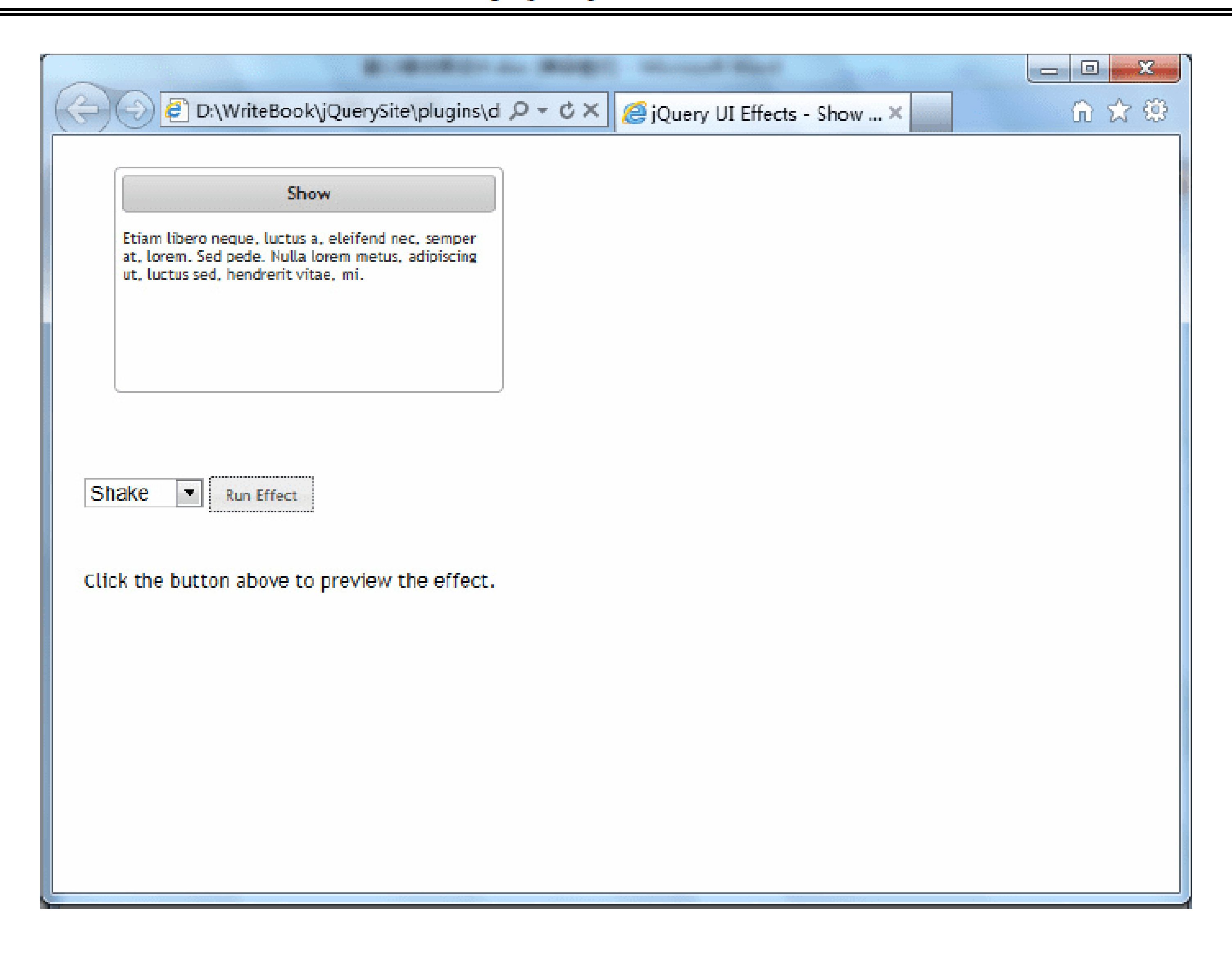


图 13.15　摆动动画效果二

13.3　小　　结

jQuery 动画效果能够给用户更愉悦的体验，也能让用户看到更生动的页面。本章主要介绍了 jQuery 动画函数，如何利用动画函数产生动画效果，以及 jQuery UI 提供的动画效果。重点内容是利用动画函数产生动画效果，jQuery UI 提供的动画效果。利用动画函数产生动画效果是本章的难点部分。下一章将介绍 jQuery 拖放组件。

13.4　习　　题

【习题 1】利用本章所学内容自定义图片的横向抖动效果。

【习题 2】利用图片练习 jQuery UI 的动画效果。

第 14 章　拖 动 插 件

现在很多网页都具有页面元素，如图片、文字段落、表格等，可以通过鼠标拖动变换位置，如博客。这种拖动效果可以通过 jQuery 的拖动插件来实现。这种拖动功能方便用户重新安排页面布局，便于用户指定符合自己风格的页面格式。本章主要讲解 jQuery 中的拖动插件。

14.1　jQuery UI 拖动插件

本节介绍 jQuery UI 工具集中的拖动插件 draggable。这个插件可以通过鼠标拖动页面元素至你想要的位置。它需要 UI Core、UI Widget、UI Mouse 库文件的支持。

14.1.1　jQuery UI 拖动插件基本介绍

jQuery UI 拖动插件的基本属性如表 14.1 所示。

表 14.1　拖动插件的属性说明

属　　性	类　　型	默　认　值	说　　明
disable	布尔	false	拖动插件是否可用
addClasses	布尔	true	是否添加新的拖动元素样式
appendTo	元素或者选择器	'parent'	添加一个元素到插件容器中
axis	字符串	false	拖动方向轴，水平和垂直两个方向
cancel	选择器	':input,option'	取消一个指定的元素拖动
connectToSortable	选择器	false	允许将一个元素拖动到可排序表格中
containment	选择器，元素，字符串，数组	false	拖动行为的页面范围，可在页面的有边界区间内拖动
cursor	字符串	'auto'	拖动时鼠标样式
cursorAt	对象	false	设定拖动元素与鼠标的相对位移
delay	整型	0	拖动动作的延迟时间，从鼠标按下开始计算
distance	整型	1	鼠标按下并移动有效距离后才开始拖动，单位为像素
grid	数组	false	捕捉拖动元素到网格
handle	元素，选择器	false	单击元素限制拖动开始
helper	字符串，函数	'original'	允许一个操作元素进行拖动显示
iFrameFix	布尔，选择器	false	在拖动过程中阻止 iFrame 捕捉鼠标移动
opacity	浮点	false	当被拖动时是否透明

续表

属　性	类　型	默 认 值	说　明
refreshPosition	布尔	false	在鼠标移动过程中即时刷新所有拖动元素的位置
revert	布尔，字符串	false	当拖动结束后元素是否返回起始位置
revertDuration	整型	500	返回起始位置的动画持续时间，单位为毫秒
scope	字符串	'default'	可拖动元素分组范围
scroll	布尔	true	当拖动时容器是否自动滚动
scrollSentivity	整型	20	可见部分的滚动距离敏感值
scrollSpeed	整型	20	窗口滚动速度，鼠标在滚动距离敏感值内
snap	布尔，选择器	false	捕捉移动边界的元素
snapMode	字符串	'both'	确定哪个边界被捕捉
snapTolerance	整型	20	捕捉到边界元素的距离
stack	选择器	false	控制层上下关系的集合
zIndex	整型	false	当元素被拖动时层的上下关系

它还具有一些拖动事件和方法，如表 14.2 和表 14.3 所示。

表 14.2　拖动插件的事件说明

事　件	类　型	说　明
create	dragcreate	拖动插件创建事件
start	dragstart	拖动行为开始事件
drag	drag	在整个拖动过程中的事件
stop	dragstop	拖动停止事件

表 14.3　拖动插件的方法说明

方　法	使 用 方 式	说　明
destroy	.draggable("destroy")	完全移除插件功能
disable	.draggable("disable")	禁用插件
enable	.draggable("enable")	启用插件
option	.draggable("option" , optionName , [value])	获取或设置插件属性
option	.draggable("option" , options)	一次设定多个插件属性
widget	.draggable("widget")	返回插件样式元素

14.1.2　jQuery UI 拖动插件示例

1. 插件的基本使用

这个示例使用了插件的属性默认值形式，具有基本的拖动功能，在插件初始化时没有为其添加任何附加特性。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable();
4   });
5   </script>
```

上述代码第 3 行使用插件初始化函数创建插件，具体效果如图 14.1 和图 14.2 所示。

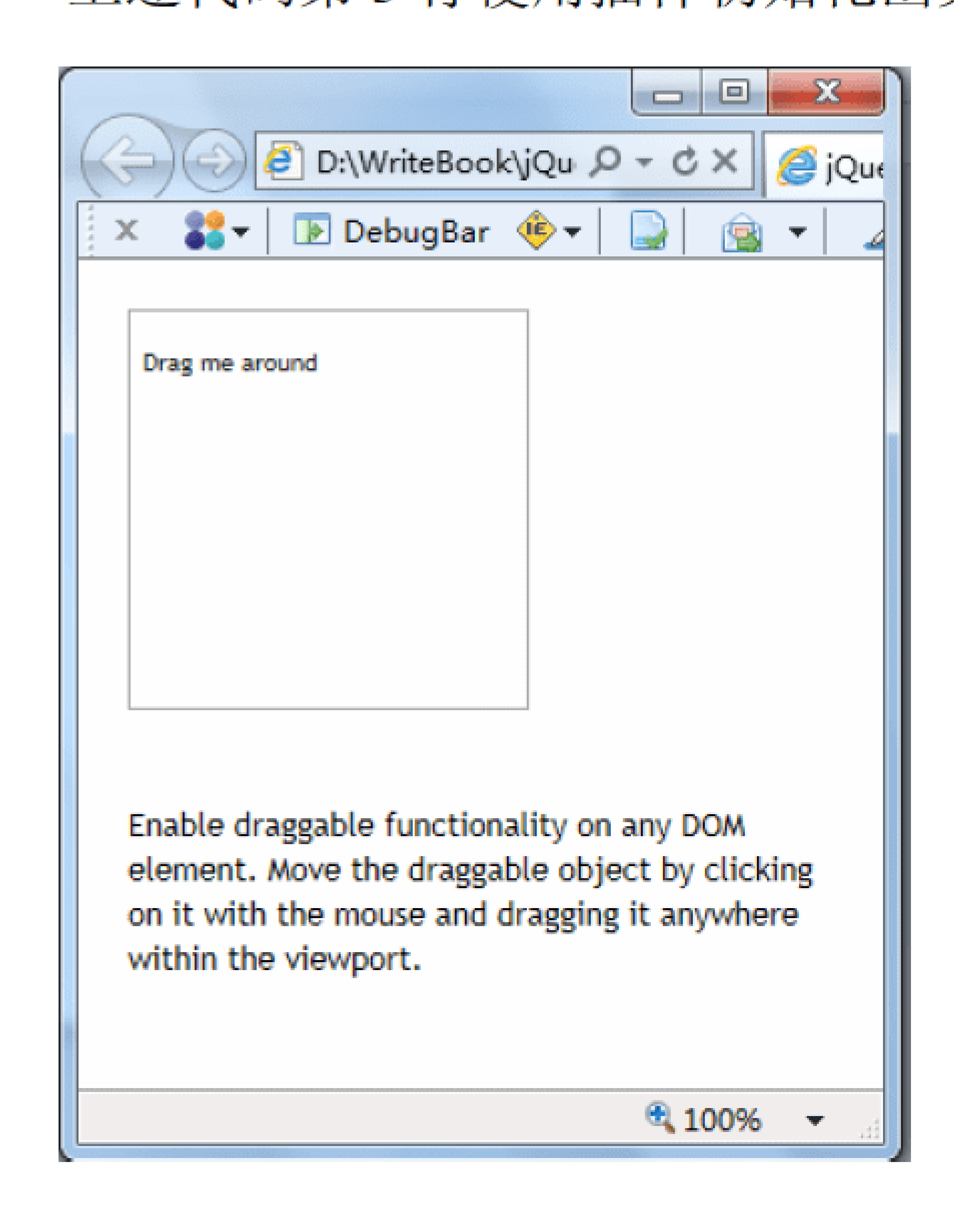



图 14.1　基本拖动应用

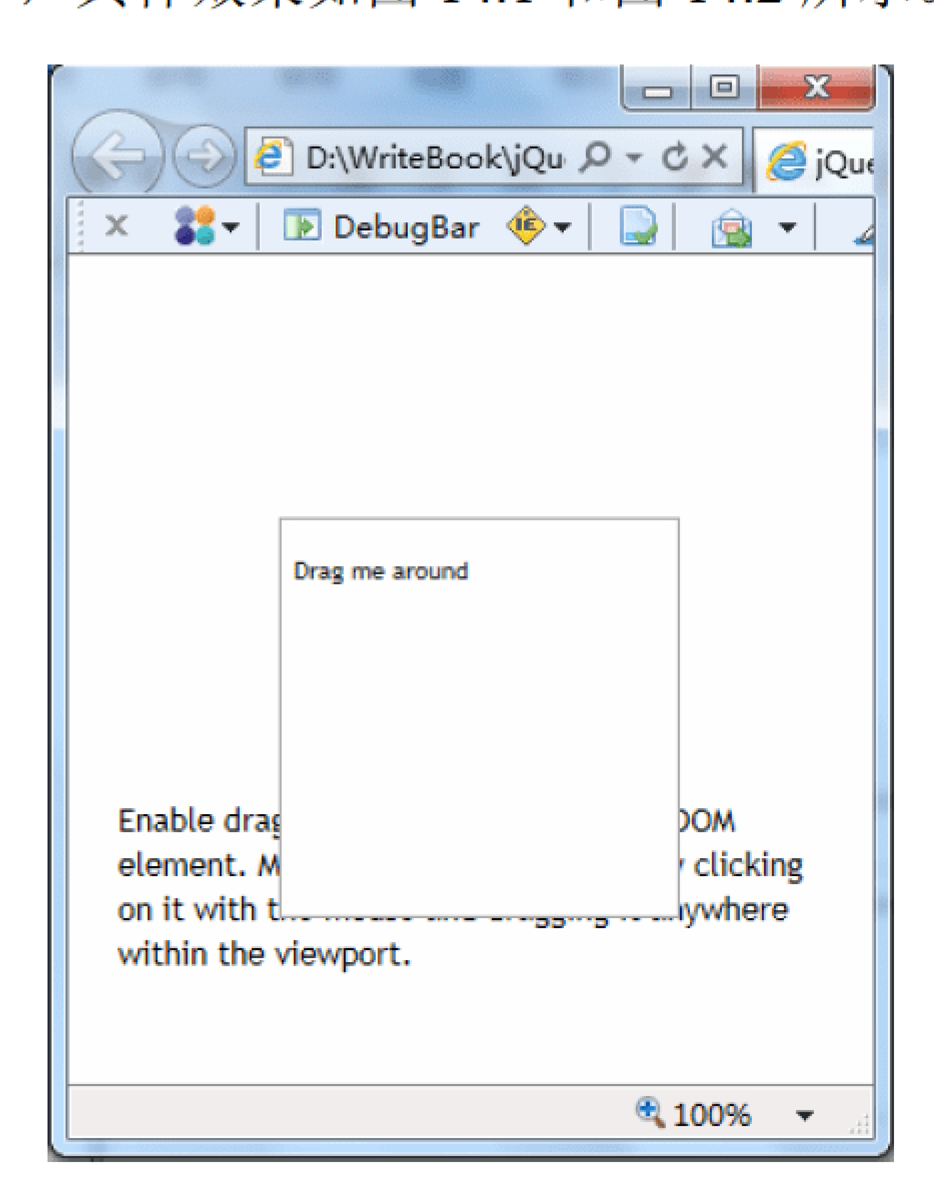


图 14.2　基本拖动应用拖动后效果

2．拖动事件的处理

在这个示例中响应了插件的开始拖动、拖动过程、拖动结束三个事件，并为事件的发生次数计数显示在拖动元素上。使用了插件的 start、drag、stop 来表示插件开始拖动、拖动中、拖动结束三个事件。并利用自定义函数 updateCounterStatus（$event_counter, new_count）记录不同事件发生的次数。

```
1   <script>
2   $(function() {
3       var $start counter = $( "#event-start" ),
4           $drag counter = $( "#event-drag" ),
5           $stop counter = $( "#event-stop" ),
6           counts = [ 0, 0, 0 ];
7       $( "#draggable" ).draggable({
8           start: function() {         //拖动行为开始事件
9               counts[ 0 ]++;
10              updateCounterStatus( $start counter, counts[ 0 ] );
11          },
12          drag: function() {          //拖动事件
13              counts[ 1 ]++;
14              updateCounterStatus( $drag counter, counts[ 1 ] );
15          },
16          stop: function() {          //拖动停止事件
17              counts[ 2 ]++;
18              updateCounterStatus( $stop counter, counts[ 2 ] );
19          }
20      });
21      function updateCounterStatus( $event counter, new count ) {
22          //第一次更新事件计数操作
```

```
23            if ( !$event_counter.hasClass( "ui-state-hover" ) ) {
24                $event_counter.addClass( "ui-state-hover" )
25                    .siblings().removeClass( "ui-state-hover" );
26            }
27            //后续事件计数操作
28            $( "span.count", $event_counter ).text( new_count );
29        }
30    });
31    </script>
```

上述代码第 3～5 行分别获取起始拖动计数、拖动计数、结束拖动计数的显示对象。第 6 行建立一个数组来记录各个事件发生的次数；第 21～29 行根据传入的不同计数显示对象和计数值更新显示。第 8～11 行响应开始拖动事件，数组元素自增并调用更新显示函数。第 12～15 行响应拖动事件，数组元素自增并调用更新显示函数。第 16～20 行响应结束拖动事件，数组元素自增并调用更新显示函数。效果如图 14.3 和图 14.4 所示。

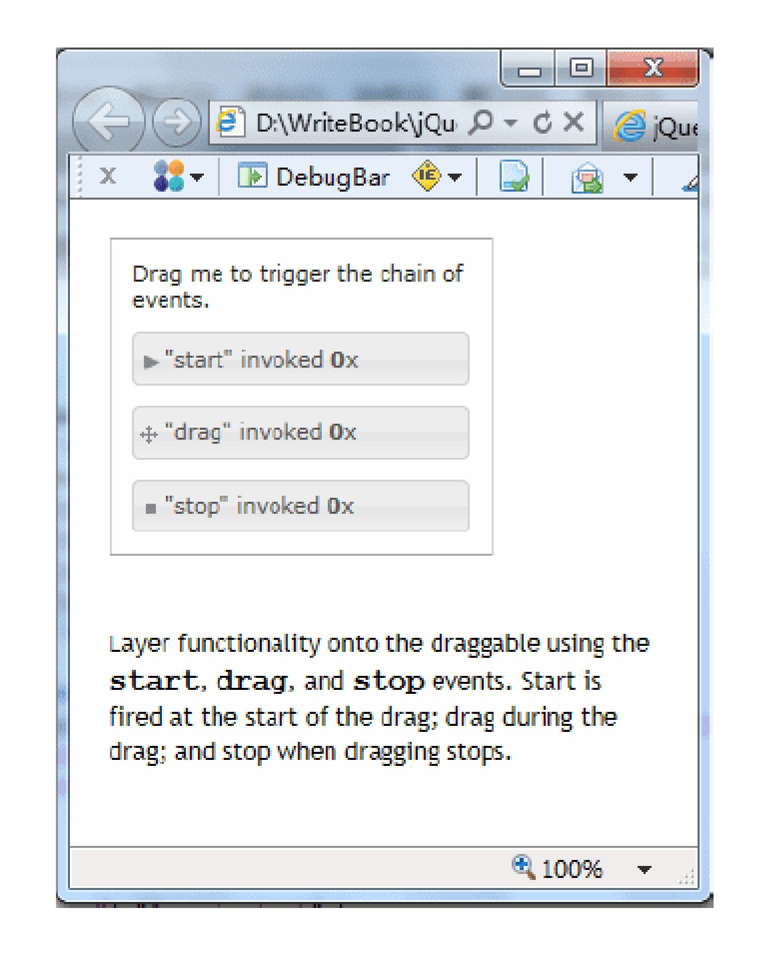


图 14.3　拖动事件处理

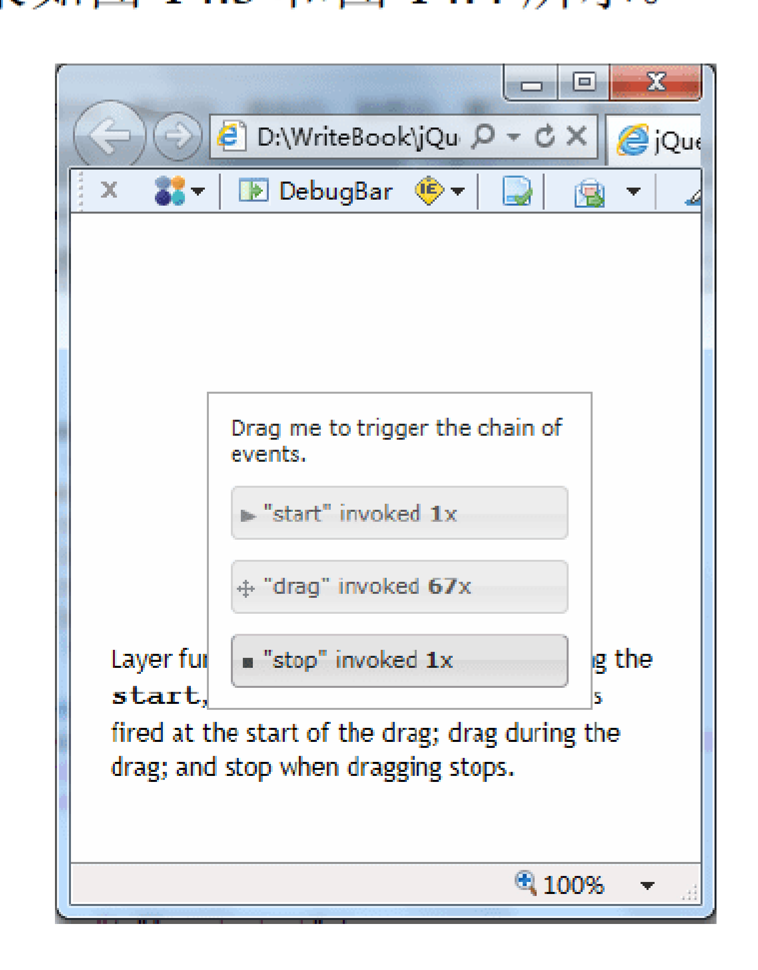


图 14.4　拖动动作发生后效果

3. 容器内拖动

这个示例使用了插件的拖动方向参数及拖动容器参数，规定拖动元素只可在某一方向上拖动，或者只可在某一容器范围内拖动。这里使用到了前面介绍的 axis 坐标轴属性及 containment 容器属性，限制了拖动方向及拖动范围。

```
1    <script>
2    $(function() {
3        $( "#draggable" ).draggable({ axis: "y" });     //纵向拖动限制
4        $( "#draggable2" ).draggable({ axis: "x" });    //横向拖动限制
5        //限定拖动范围
6        $( "#draggable3" ).draggable({ containment: "#containment-wrapper",
         scroll: false });
7        $( "#draggable4" ).draggable({ containment: "#demo-frame" });
8        $( "#draggable5" ).draggable({ containment: "parent" });
9    });
10   </script>
```

上述代码第 3 行规定拖动元素只可在纵向拖动。第 4 行规定拖动元素只可在横向拖动。第 6 行规定拖动元素只可在指定容器内拖动，并且不可使用滚动条。第 7 行规定拖动元素只可在指定容器内拖动，但页面上并无此容器，使用滚动条。第 8 行规定拖动元素只可在父元素中拖动。效果如图 14.5 和图 14.6 所示。

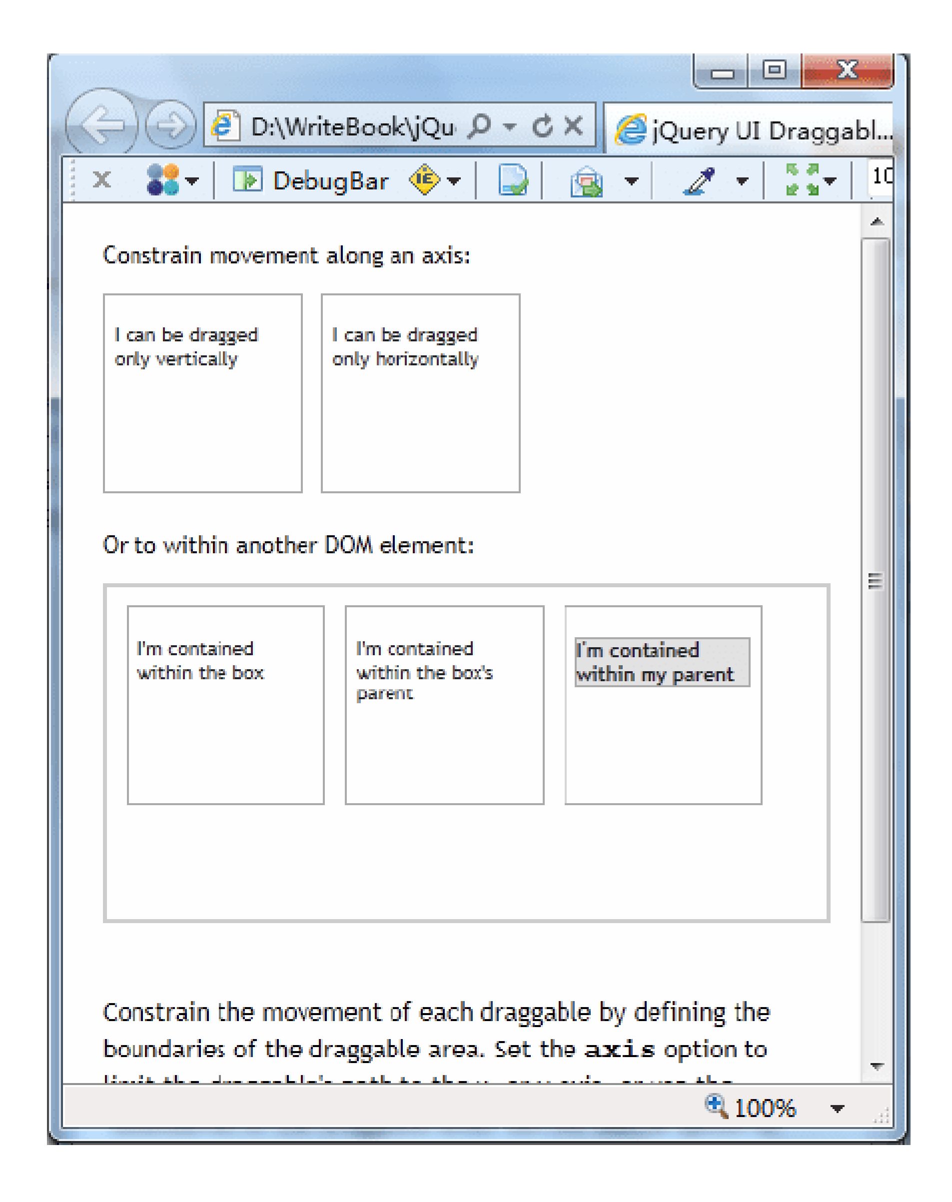


图 14.5　容器内拖动

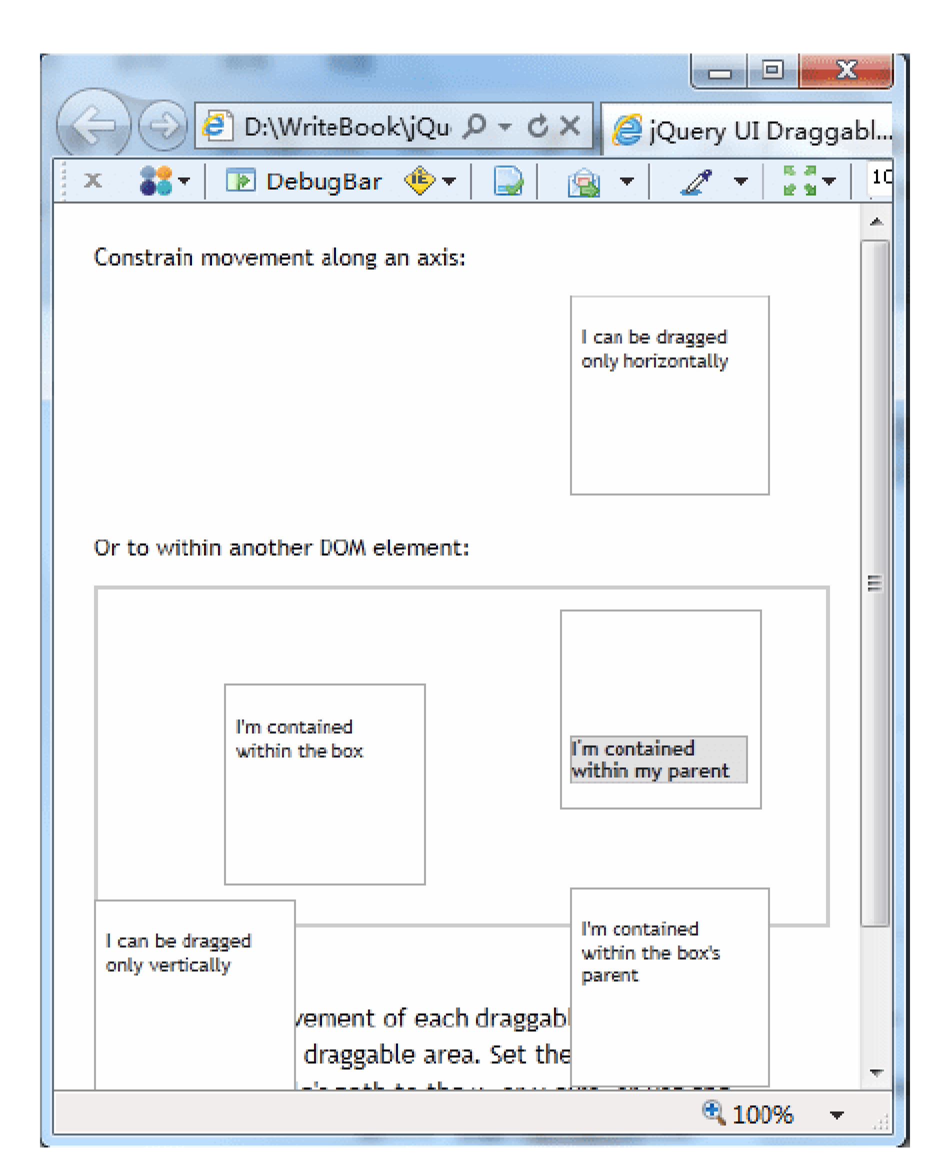


图 14.6　容器内拖动后效果

4．延迟拖动

这个示例使用了插件在距离和时间上的延迟拖动效果，避免出现用户频繁单击出现的误操作效果。这里使用了前面介绍的 distance 延迟移动距离和 delay 延迟毫秒数属性。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable({ distance: 20 });
                                        //设定拖动时鼠标反应距离
4       $( "#draggable2" ).draggable({ delay: 1000 }); //设定拖动延迟时间
5       $( ".ui-draggable" ).disableSelection();
6   });
7   </script>
```

上述代码第 3 行设定当鼠标按下并移动了 20 像素后再发生拖动操作。第 4 行设定当鼠标按下并移动 1 秒钟后再发生拖动操作。第 5 行禁止样式选定。效果如图 14.7 和图 14.8 所示。

5．捕获至元素或者网格

这个示例使用了拖动插件的捕获属性。这里使用了前面介绍的 snap 捕获属性和 grid 网格属性。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable({ snap: true });   //拖动元素捕获设定
4       $( "#draggable2" ).draggable({ snap: ".ui-widget-header" });
```

```
                                          //设定捕获拖动元素对象
5       $("#draggable3").draggable({ snap: ".ui-widget-header", snapMode:
        "outer" });                       //设定捕获拖动元素对象及捕获方式
6       $( "#draggable4" ).draggable({ grid: [ 20,20 ] });
7       $( "#draggable5" ).draggable({ grid: [ 80, 80 ] });
8    });
9    </script>
```

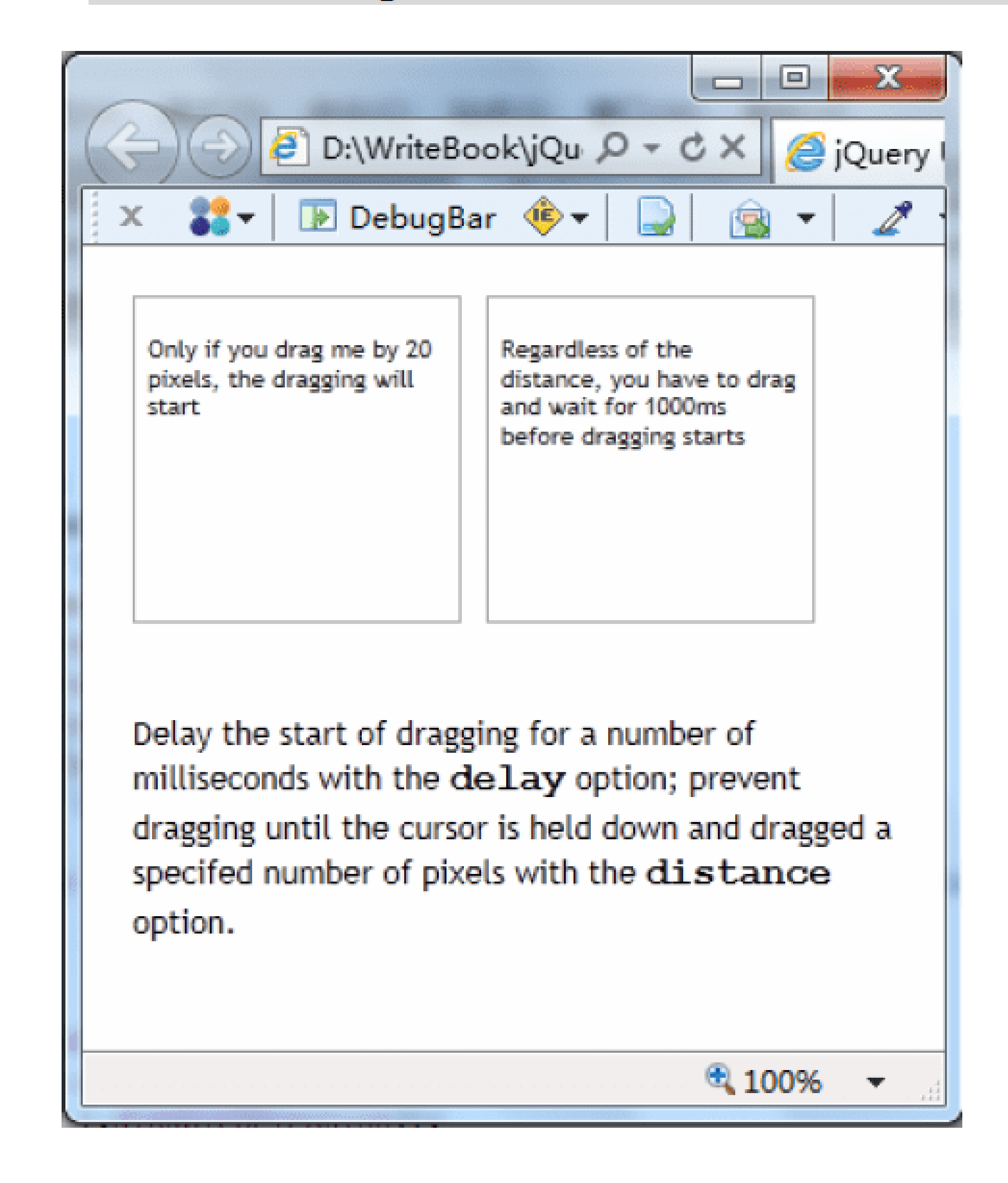


图 14.7　延迟拖动

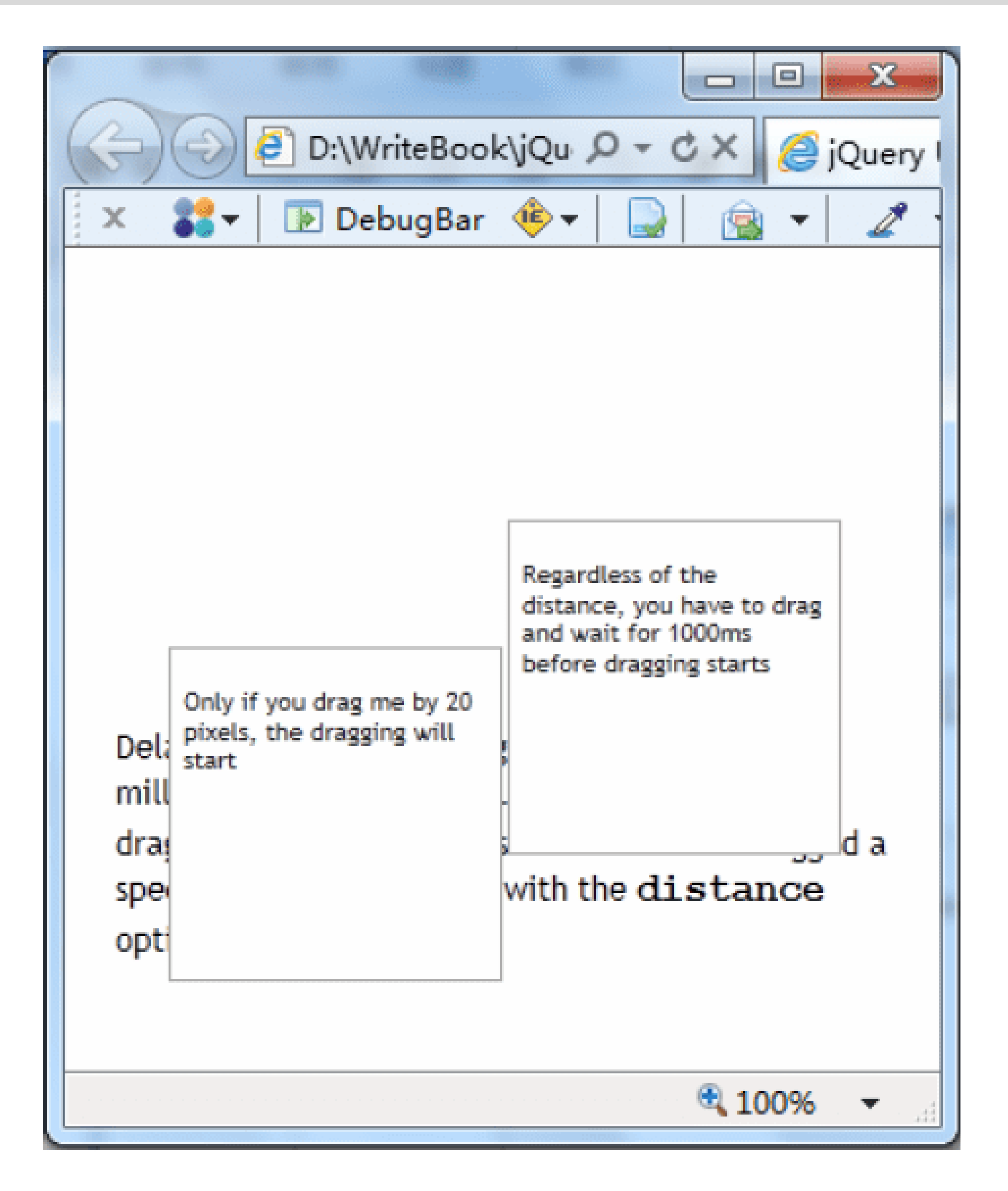


图 14.8　延迟拖动后结果

上述代码第 3 行简单设置了拖动元素可被捕获。第 4 行指定了被拖动元素被哪个元素捕获。第 5 行也是指定了捕获元素及捕获方式。第 6、7 行指定了拖动元素被捕获的网格位置。效果如图 14.9 和图 14.10 所示。

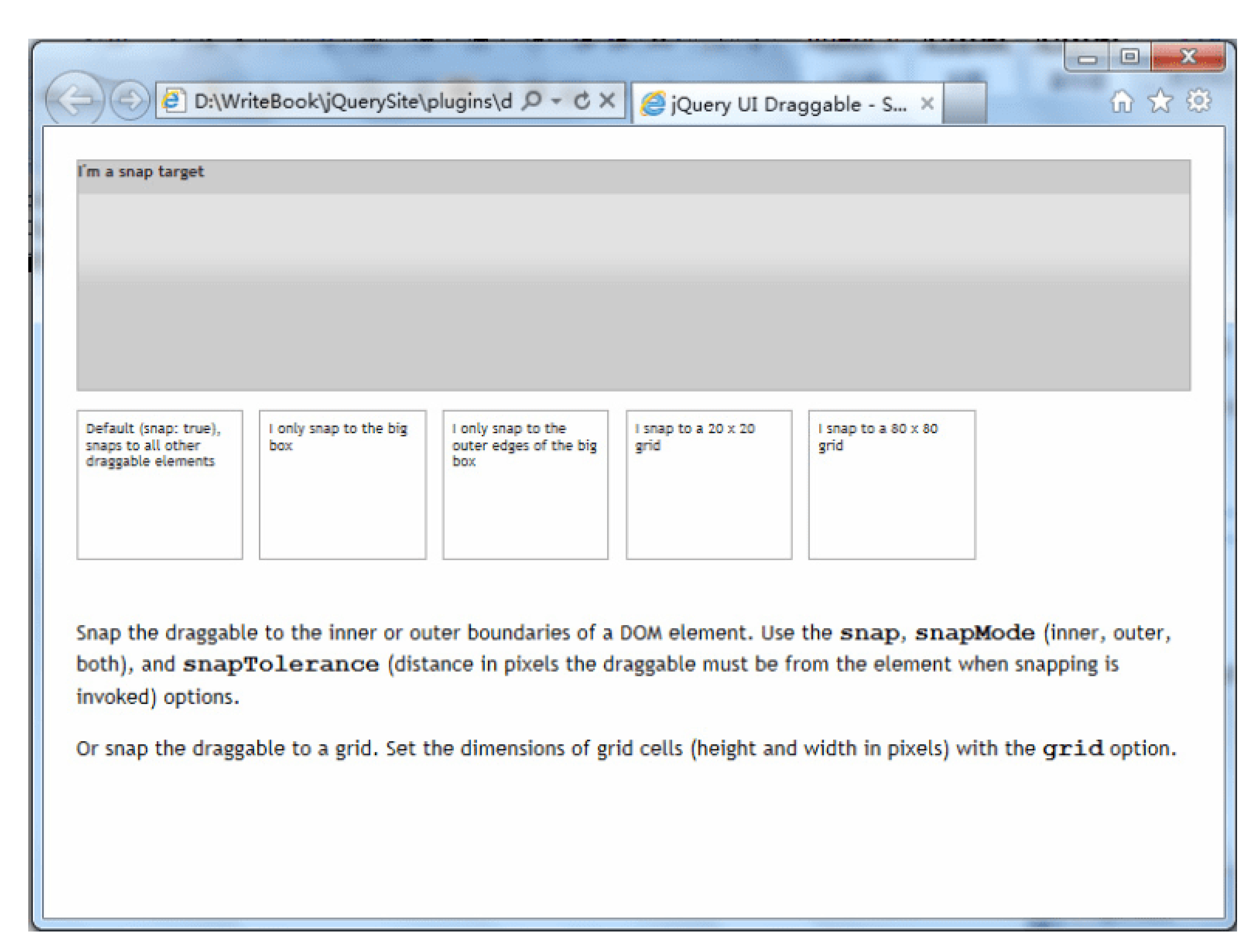


图 14.9　捕获至元素或网格

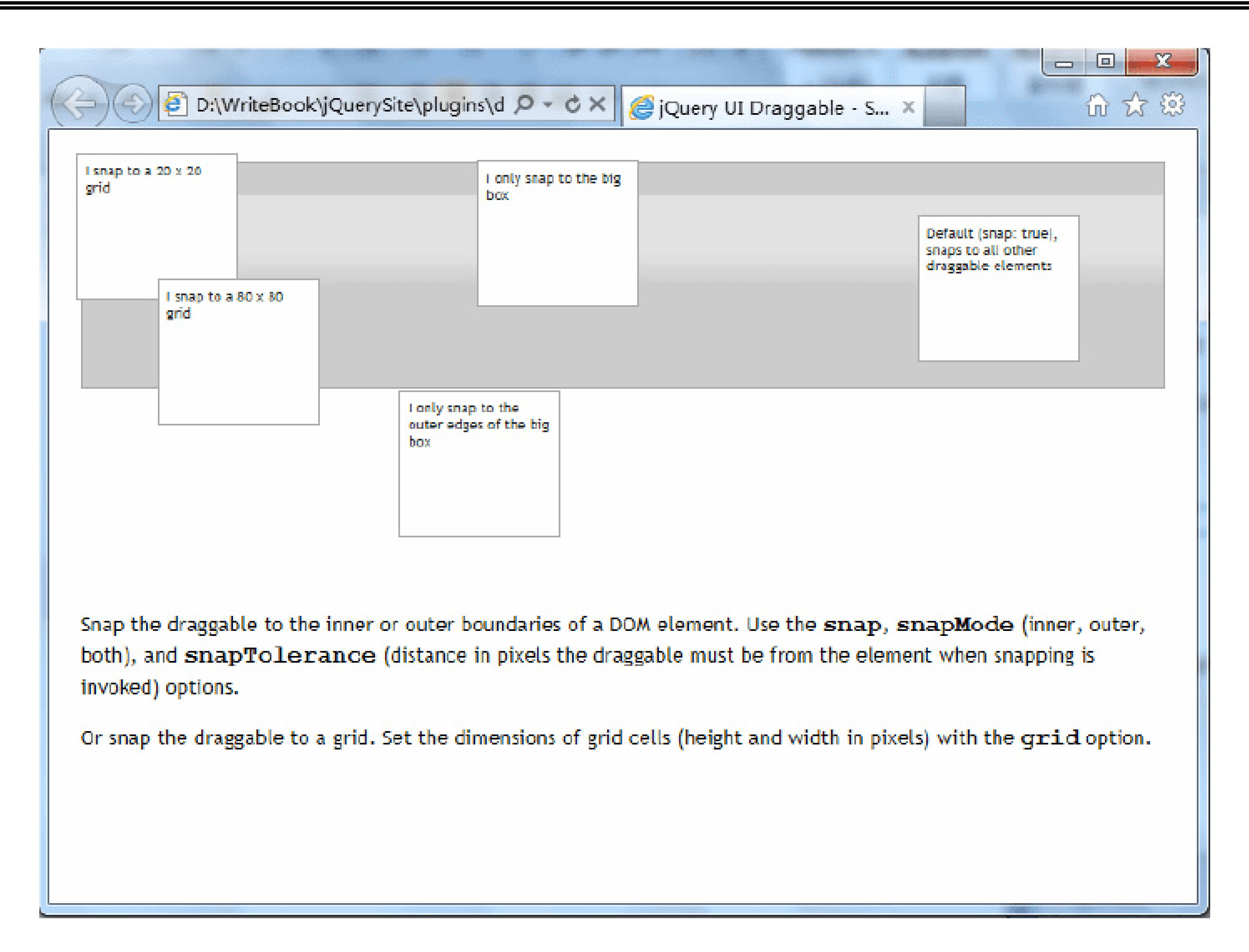


图 14.10　被元素或网格捕获效果

6．拖动过程中窗口自动滚动

这个示例使用了与滚动条相关的三个属性：scroll，是否添加滚动条，scrollSensitiyity，滚动条敏感值，scrollSpeed，滚动条的滚动速度。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable({ scroll: true });
                                              //拖动中父容器可自动产生滚动条
4       $( "#draggable2" ).draggable({ scroll: true, scrollSensitivity:
        100 });                               //滚动条的敏感值
5       $( "#draggable3" ).draggable({ scroll: true, scrollSpeed: 100 });
                                              //滚动条的滚动速度
6   });
7   </script>
```

上述代码第 3 行设定了窗口可随拖动操作自动滚动。第 4 行设定了窗口自动滚动的敏感距离。第 5 行设定了自动滚动速度。效果如图 14.11 和图 14.12 所示。

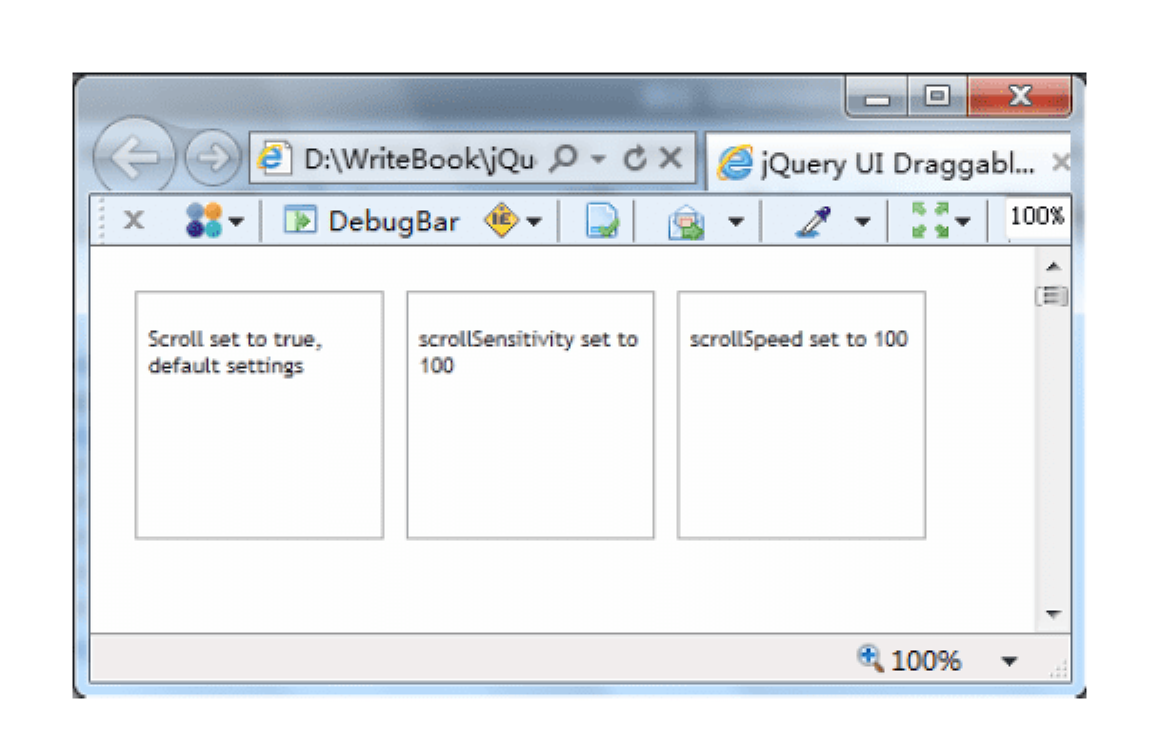


图 14.11　自动滚动窗口

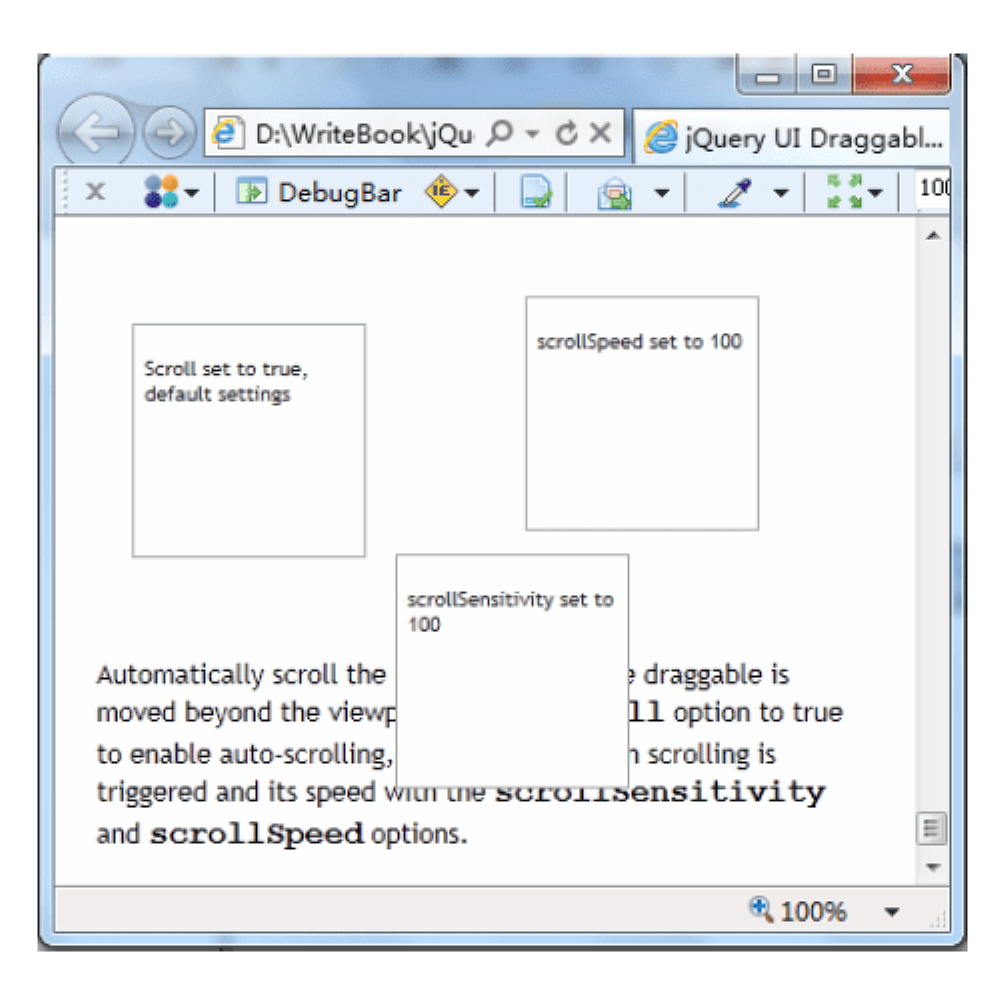


图 14.12　拖动引发窗口自动滚动

7. 恢复原始位置

这个示例使用了插件的恢复位置属性。这里使用了插件的 revert 恢复原始位置属性。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable({ revert: true }); //拖动后返回原始位置
4       $( "#draggable2" ).draggable({ revert: true, helper: "clone" });
                                                        //拖动后产生克隆元素
5   });
6   </script>
```

上述代码第 3 行简单设置了拖动元素的恢复原始位置属性。第 4 行设定了拖动元素在被拖动时，插件工具自动在原位置创建的克隆体，并恢复原始位置。这个例子的效果是，拖动元素还可拖动，但当我们将鼠标键抬起时，元素又恢复到了初始位置。效果如图 14.13～图 14.15 所示。

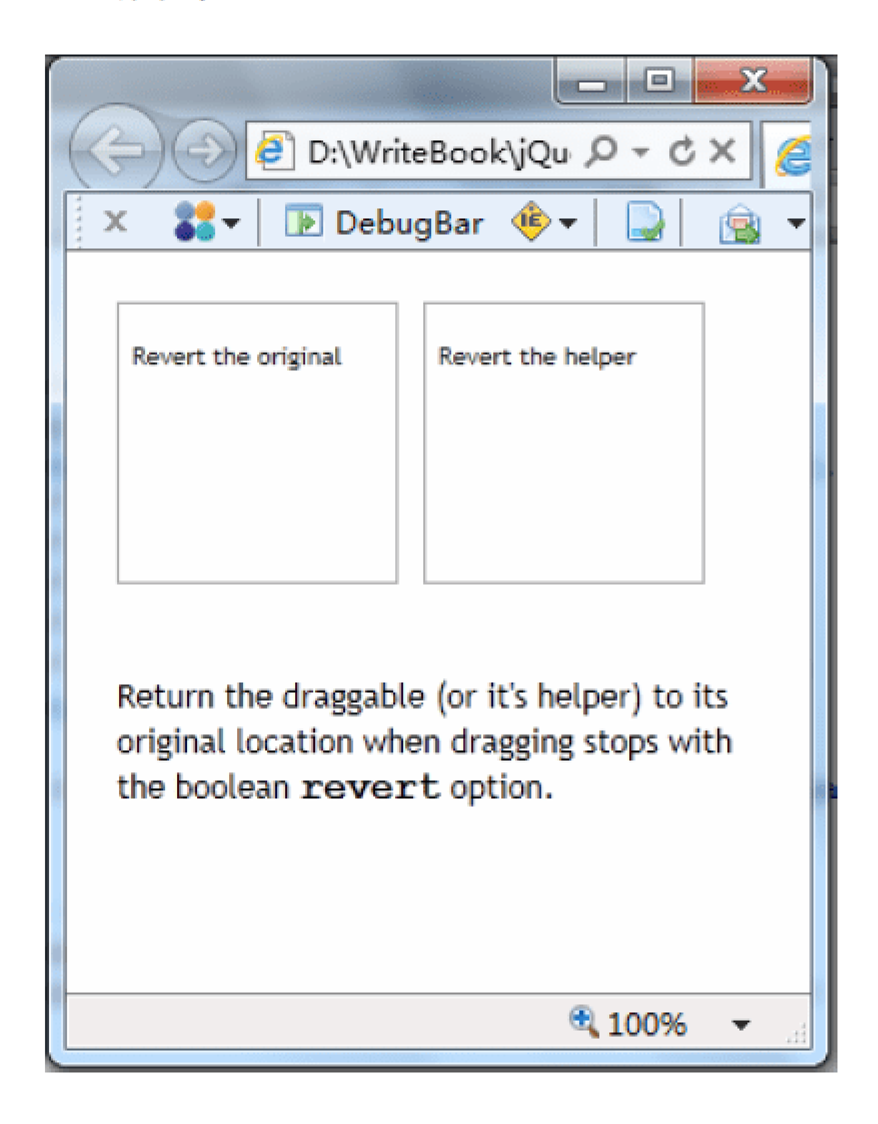


图 14.13　恢复原始位置

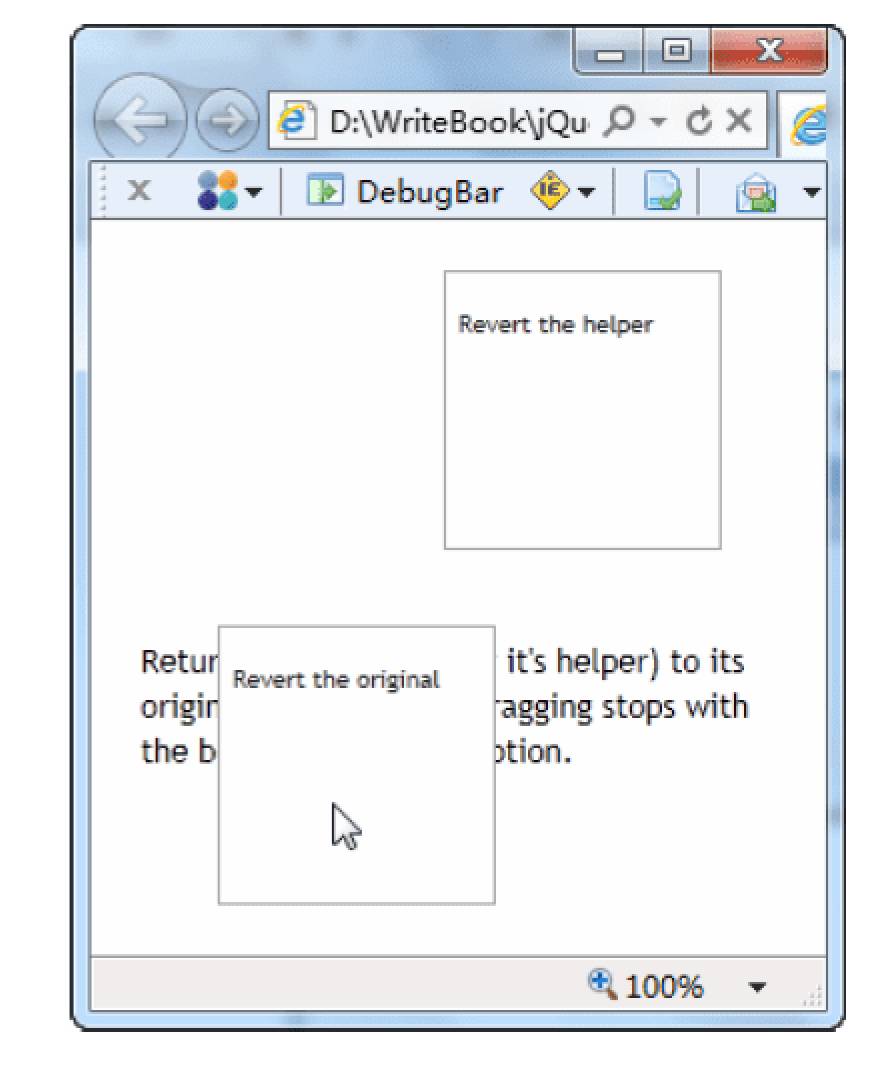


图 14.14　恢复原始位置拖动过程一

8. 可视化反馈

这个示例使用了插件的工具设定，使用插件工具在拖动时实现了不同效果，另外还使用了拖动元素的堆栈。这里使用了插件的 opacity 透明度属性，以及当拖动过程中设置鼠标样式的属性 cursor 和鼠标相对于拖动元素的位移 cursorAt 属性。使用 stack 堆栈属性设定了在同一个栈中的不同拖动元素当被拖动产生覆盖效果时，最后一次拖动的元素在其他元素的上层，并覆盖了其他被拖动的元素。

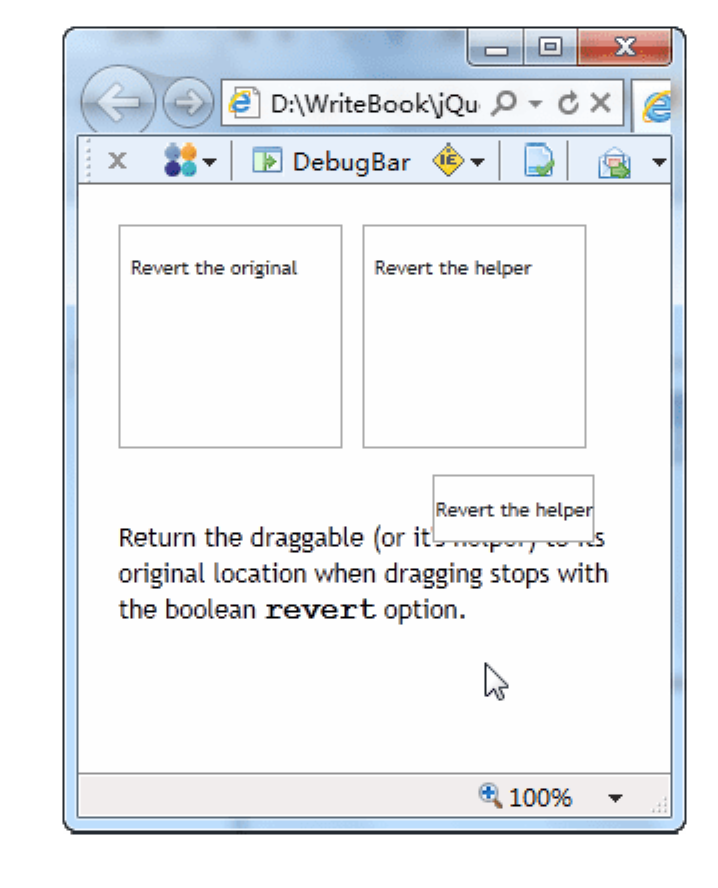


图 14.15　恢复原始位置拖动过程二

```
1   <script>
2   $(function() {
```

```
3       $( "#draggable" ).draggable({ helper: "original" });
4       $( "#draggable2" ).draggable({ opacity: 0.7, helper: "clone" });
5       $( "#draggable3" ).draggable({
6           cursor: "move",                     //鼠标样式
7           cursorAt: { top: -12, left: -20 },   //鼠标位置
8           helper: function( event ) {          //对拖动元素产生新内容
9               return $( "<div class='ui-widget-header'>I'm a custom helper
                </div>" );
10          }
11      });
12      $( "#set div" ).draggable({ stack: "#set div" });
13  });
14  </script>
```

上述代码第 3 行使用插件原始工具功能。第 4 行设定当拖动元素时，元素透明度改变，并使用克隆工具克隆元素在原有位置。第 6 行设定拖动时光标样式。第 7 行设定拖动时光标位置。第 8、9 行重写了工具的实现功能，用一个新添加的层表示拖动效果。第 12 行使用拖动元素堆栈，效果如图 14.16～图 14.18 所示。

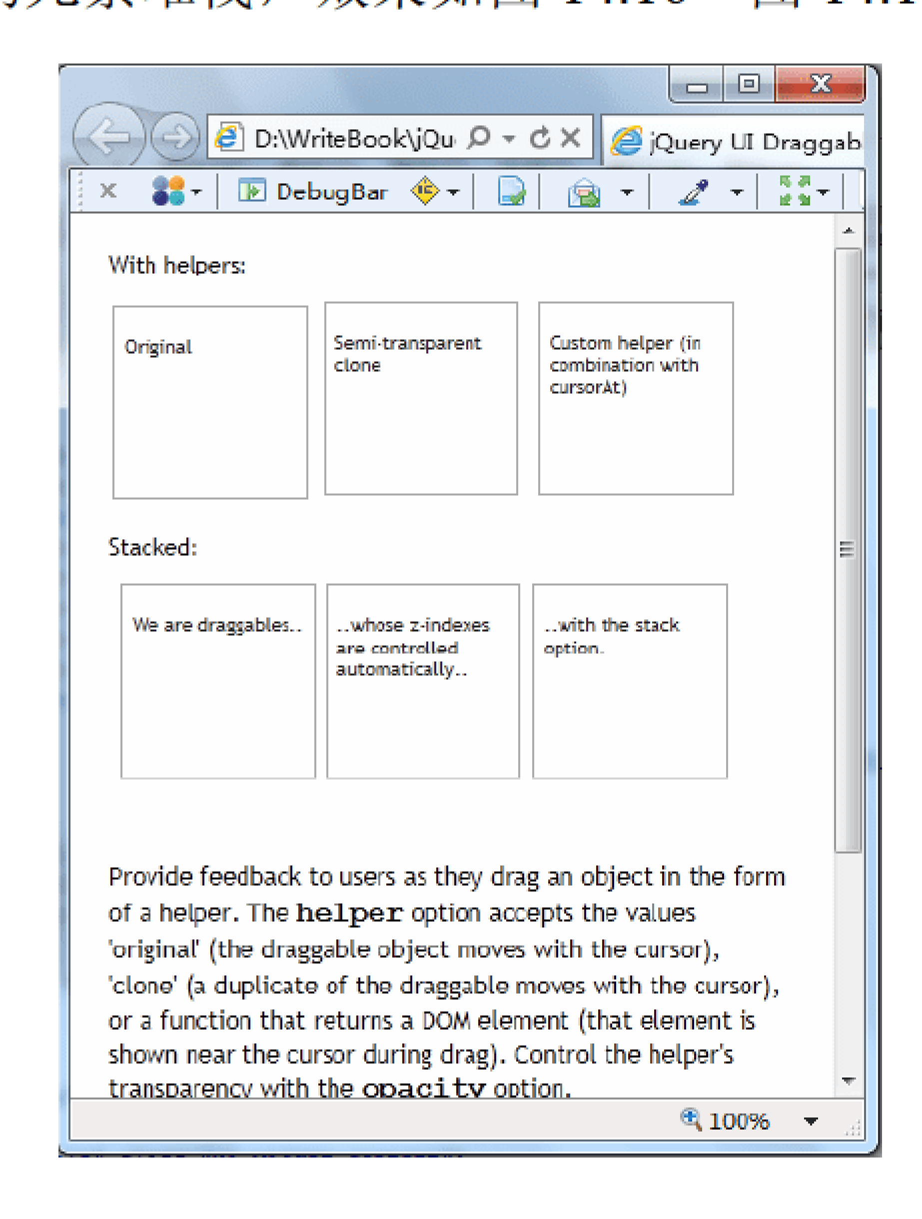


图 14.16　可视化反馈

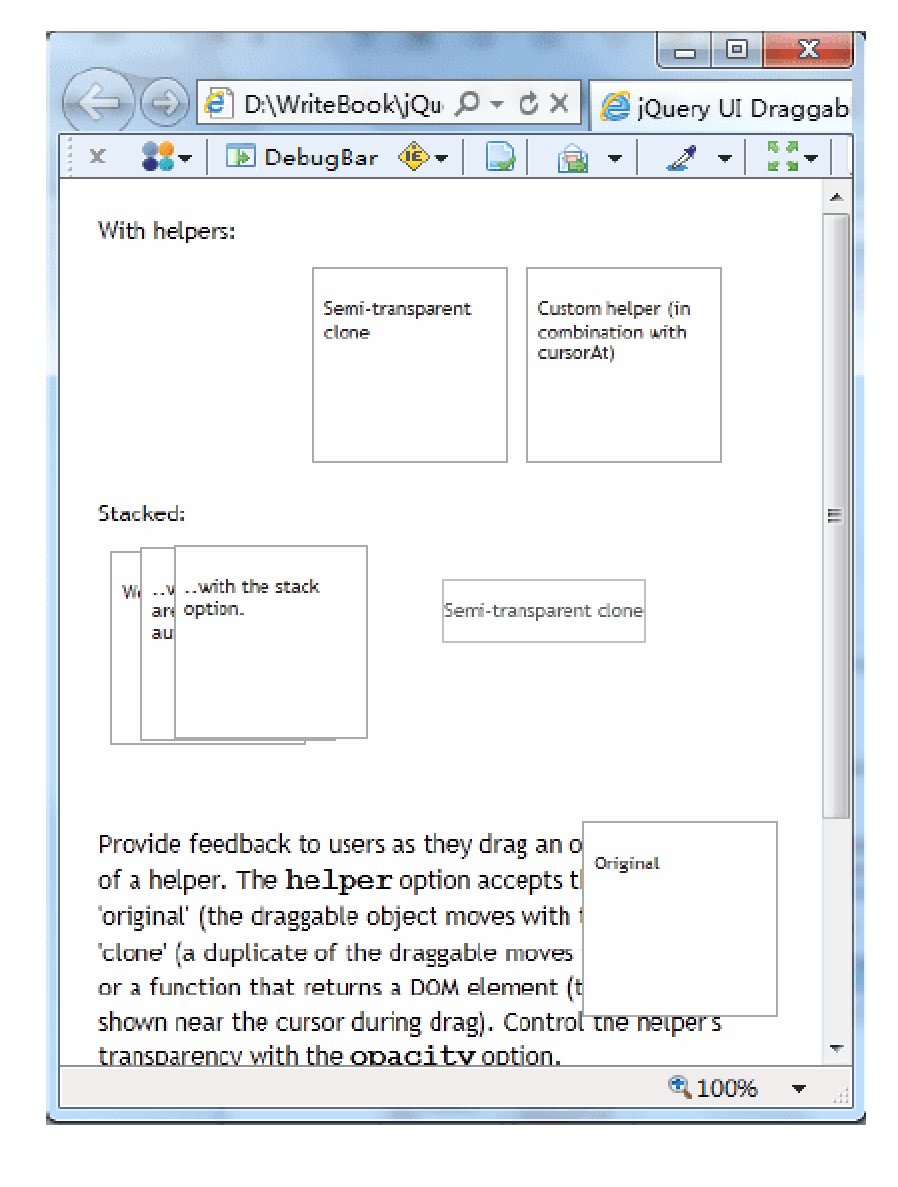


图 14.17　可视化反馈透明度 0.7 效果

9．光标样式

这个示例使用了插件对光标设置的相关属性。这里使用了和上一个例子相同的关于光标样式设定的两个属性 cursor 和 cursorAt。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable({ cursorAt: { cursor: "move", top: 56,
        left: 56 } });
4       $("#draggable2").draggable({ cursorAt:{ cursor:"crosshair", top:
        -5, left: -5 } });
5       $( "#draggable3" ).draggable({ cursorAt: { bottom: 0 } });
```

```
6    });
7    </script>
```

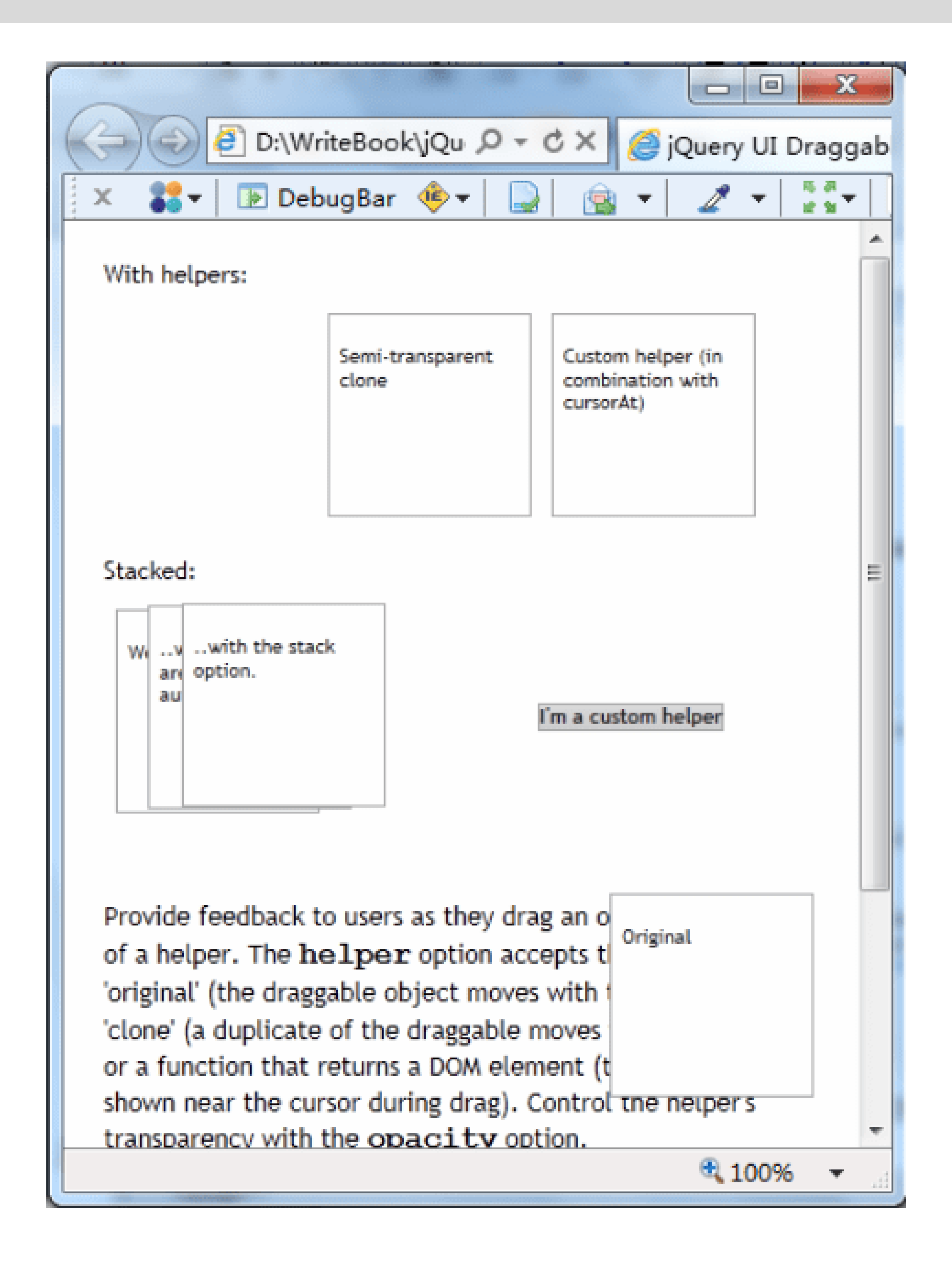


图 14.18　可视化反馈重写工具实现

上述代码第 3 行设定了光标位置，光标样式为移动样式。第 4 行设定了光标位置，光标样式为十字样式。第 5 行设定了光标位置在拖动元素底端。效果如图 14.19～图 14.21 所示。

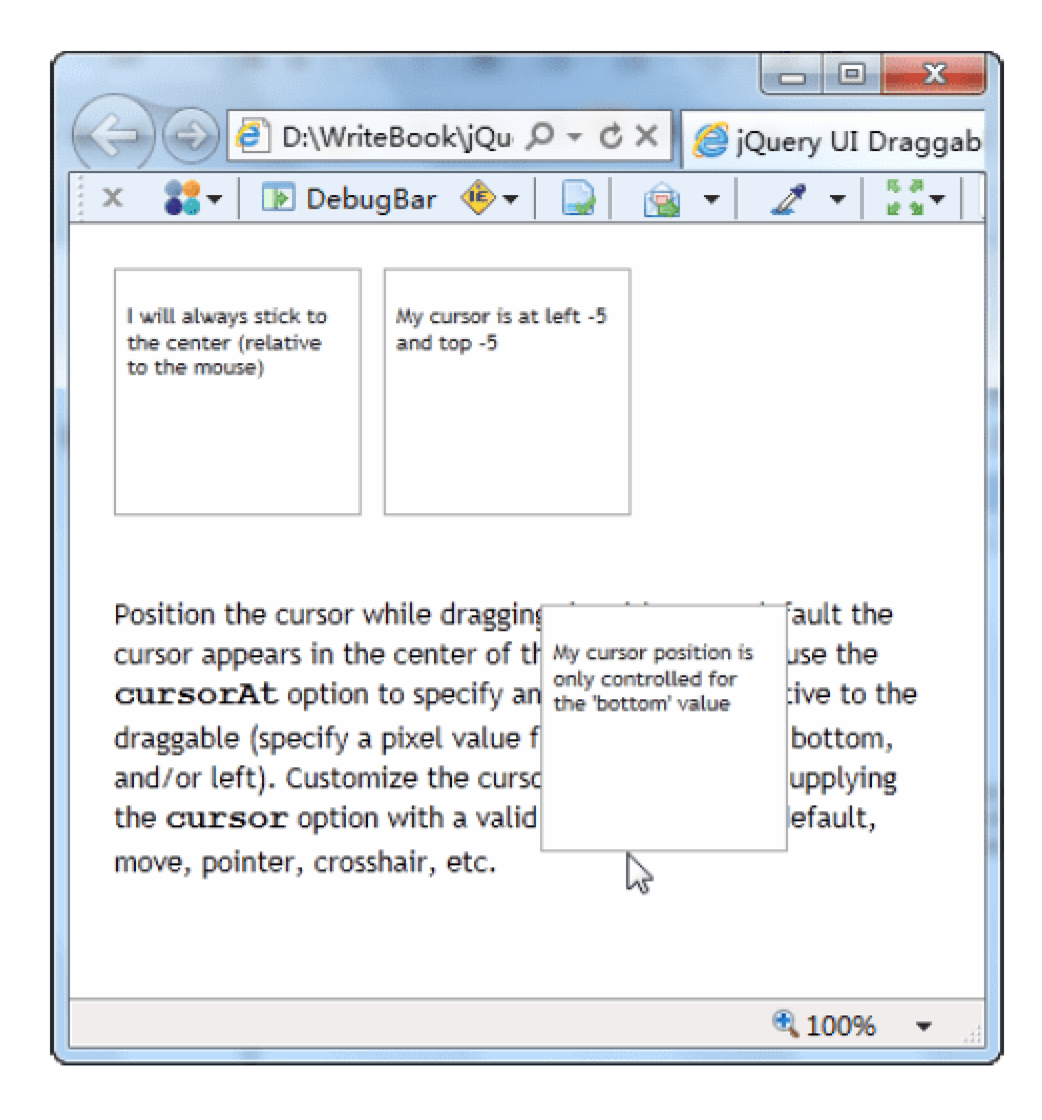




图 14.19　光标样式一

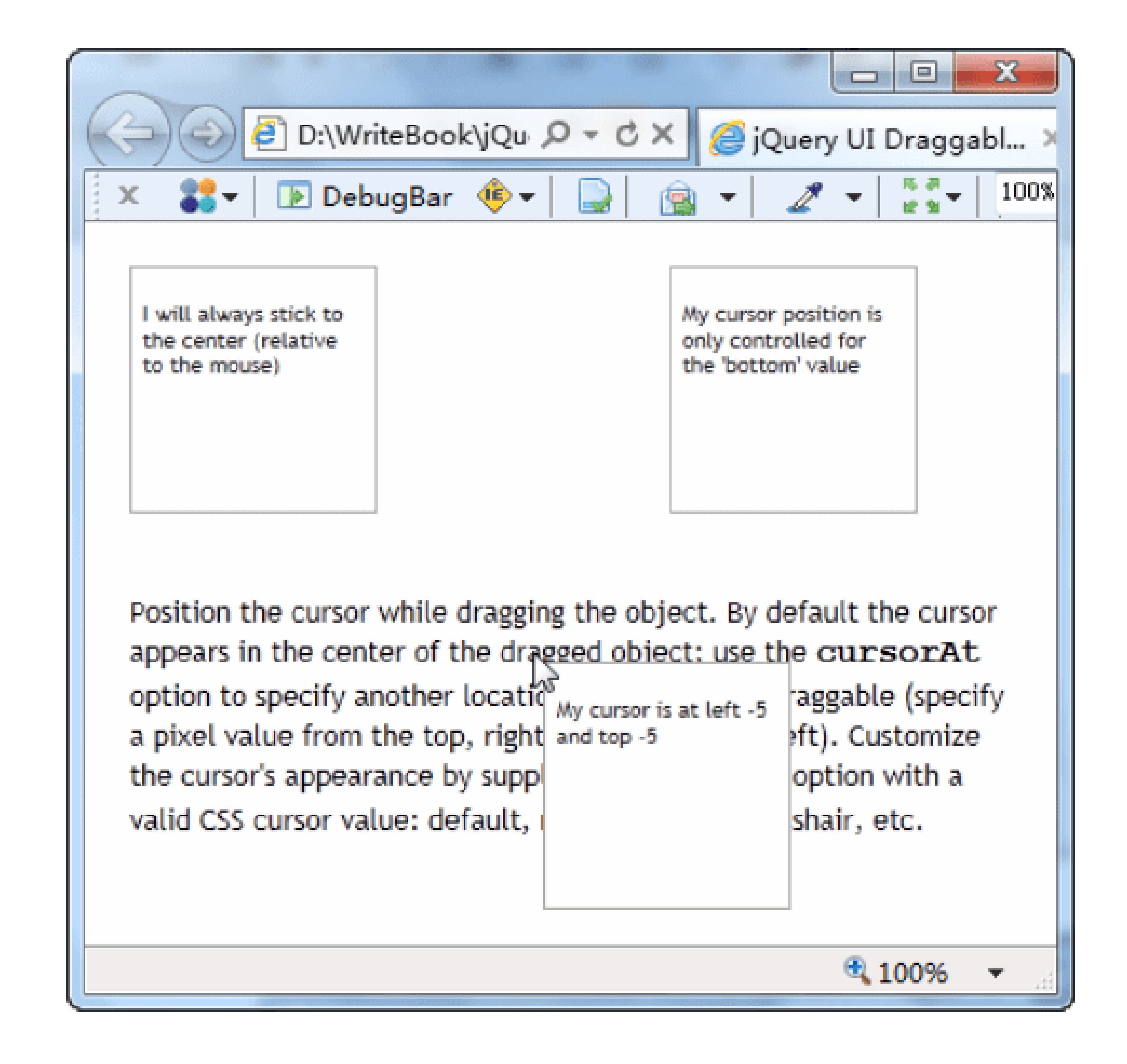


图 14.20　光标样式二

10．拖入有序表格

这个示例使用了排序列表插件，将拖动元素拖入排序列表中。这里使用了可排序列表插件 sortable，并设定了表中元素定位属性 revert。在拖动插件中使用了 connectToSortable 属性与排序表格关联。

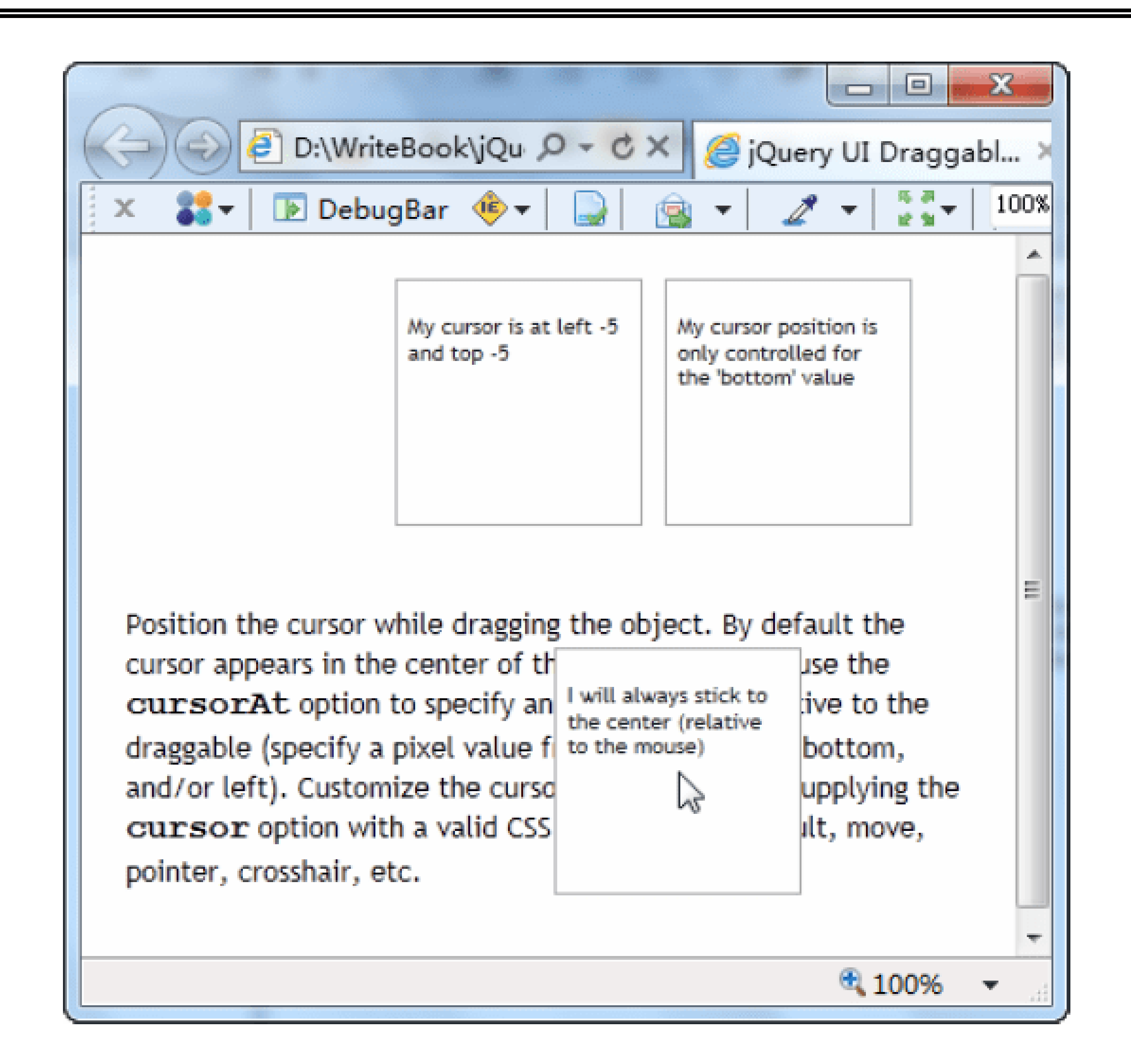


图 14.21　光标样式三

```
1   <script>
2   $(function() {
3       $( "#sortable" ).sortable({
4           revert: true
5       });
6       $( "#draggable" ).draggable({
7           connectToSortable: "#sortable",     //将拖动元素与表格关联
8           helper: "clone",
9           revert: "invalid"
10      });
11      $( "ul, li" ).disableSelection();
12  });
13  </script>
```

上述代码第 4 行设定排序表格行顺序发生变化时元素定位在新的位置。第 7 行将拖动插件与排序表格关联起来。第 8 行设定当拖动元素时克隆一个新的元素。第 9 行不允许元素恢复原始位置。效果如图 14.22 和图 14.23 所示。

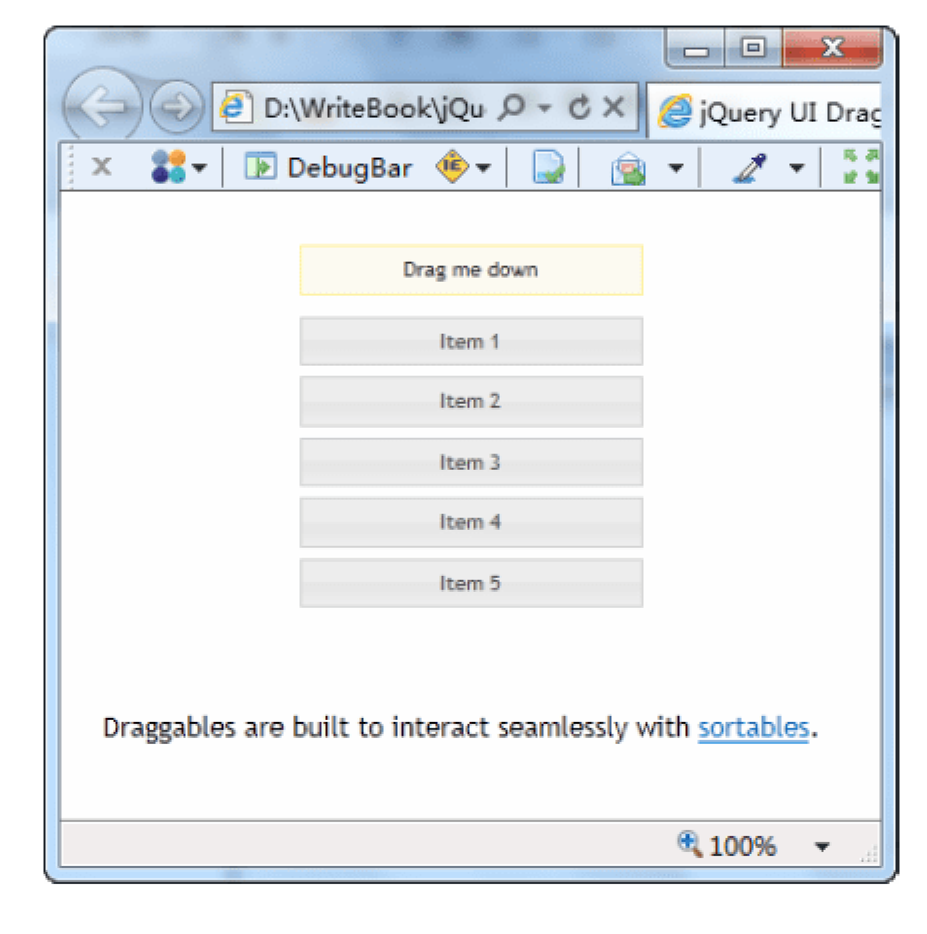


图 14.22　拖入有序表格一

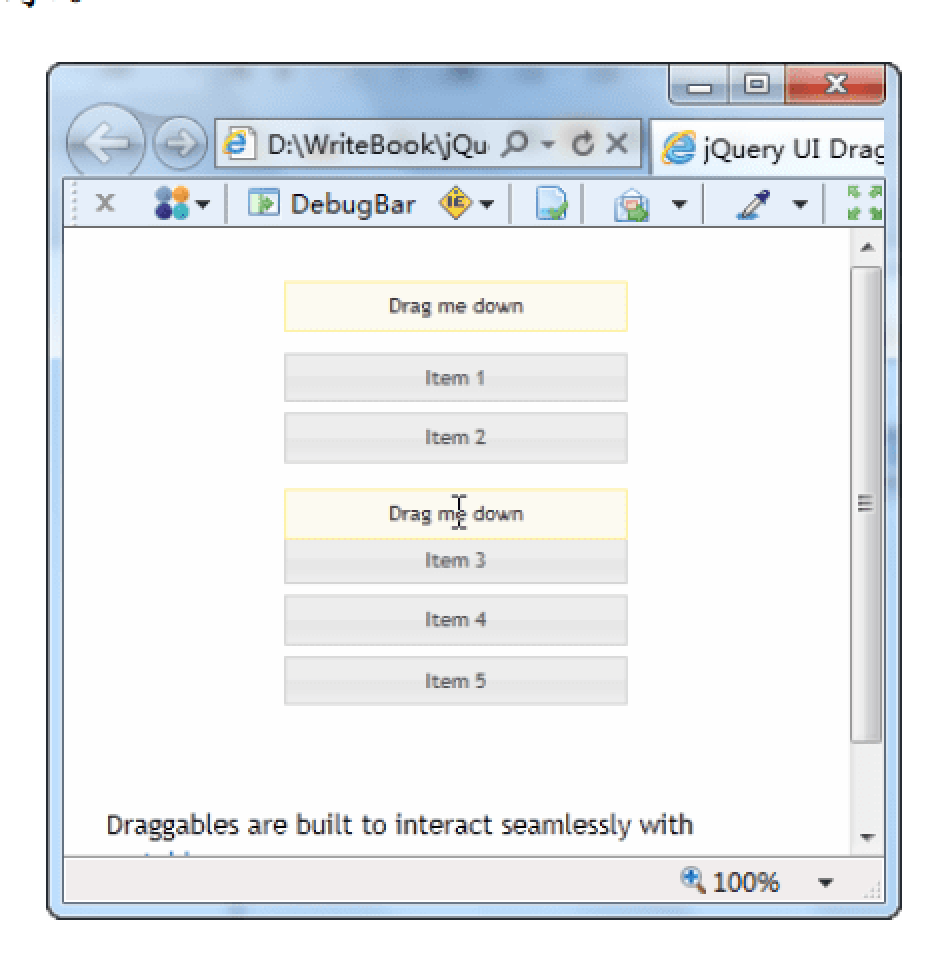


图 14.23　拖入有序表格二

14.2　jQuery UI 投放插件

下面介绍 jQuery UI 提供的另一种投放插件 dropable。这种插件可以与上一节讲的拖动插件搭配使用，不同的是这种插件可以接受拖动插件元素。

14.2.1　jQuery UI 投放插件基本介绍

下面来看一下 jQuery UI 投放插件的基本属性，如表 14.4 所示。

表 14.4　投放插件属性说明

属　性	类　型	默 认 值	说　明
disable	布尔	false	是否使用投放插件
accept	选择器，函数	'*'	所有的匹配选择器的拖动对象都会被接受
activeClass	字符串	false	投放元素的激活样式
addClasses	布尔	true	是否添加 ui-droppable 样式
hoverClass	字符串	false	当拖动元素悬停在投放元素上时的样式类
greedy	布尔	false	是否阻止嵌套的投放元素事件传递到上层元素
scope	字符串	'default'	投放元素分组范围
tolerance	字符串	'intersect'	指定当拖动元素覆盖在投放元素上时，投放元素的样式

该插件还具有一些投放事件和方法，如表 14.5 和表 14.6 所示。

表 14.5　投放插件事件说明

事　件	类　型	说　明
create	dropcreate	投放控件创建事件
activate	dropactivate	投放元素激活事件
deactivate	dropdeactivate	投放元素停止移动事件
over	dropover	拖动元素悬停在投放元素上事件
out	dropout	拖动元素离开投放元素事件
drop	drop	拖动元素被拖动到投放元素上事件

表 14.6　投放插件方法说明

方　法	使 用 方 式	说　明
destroy()	.droppable("destroy")	移除插件所有功能
disable()	.droppable("disable")	禁用插件
enable()	.droppable("enable")	启用插件
option()	.droppable("option" , optionName , [value])	设定插件选项
widget()	.droppable("widget")	获取插件组件

14.2.2　jQuery UI 投放插件示例

1．默认功能

这个示例使用了投放的默认属性功能。这里还使用了插件的投放事件 drop，并在投放

事件中修改投放插件的样式及文字内容。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable();
4       $( "#droppable" ).droppable({
5           drop: function( event, ui ) {       //投放事件
6               $( this )
7                   .addClass( "ui-state-highlight" )
8                   .find( "p" )
9                       .html( "Dropped!" );
10          }
11      });
12  });
13  </script>
```

上述代码第 3 行初始化一个拖动插件。第 4 行初始化一个投放插件。第 5～10 行响应投放插件的投放事件，在事件中修改样式及文本。效果如图 14.24 和图 14.25 所示。

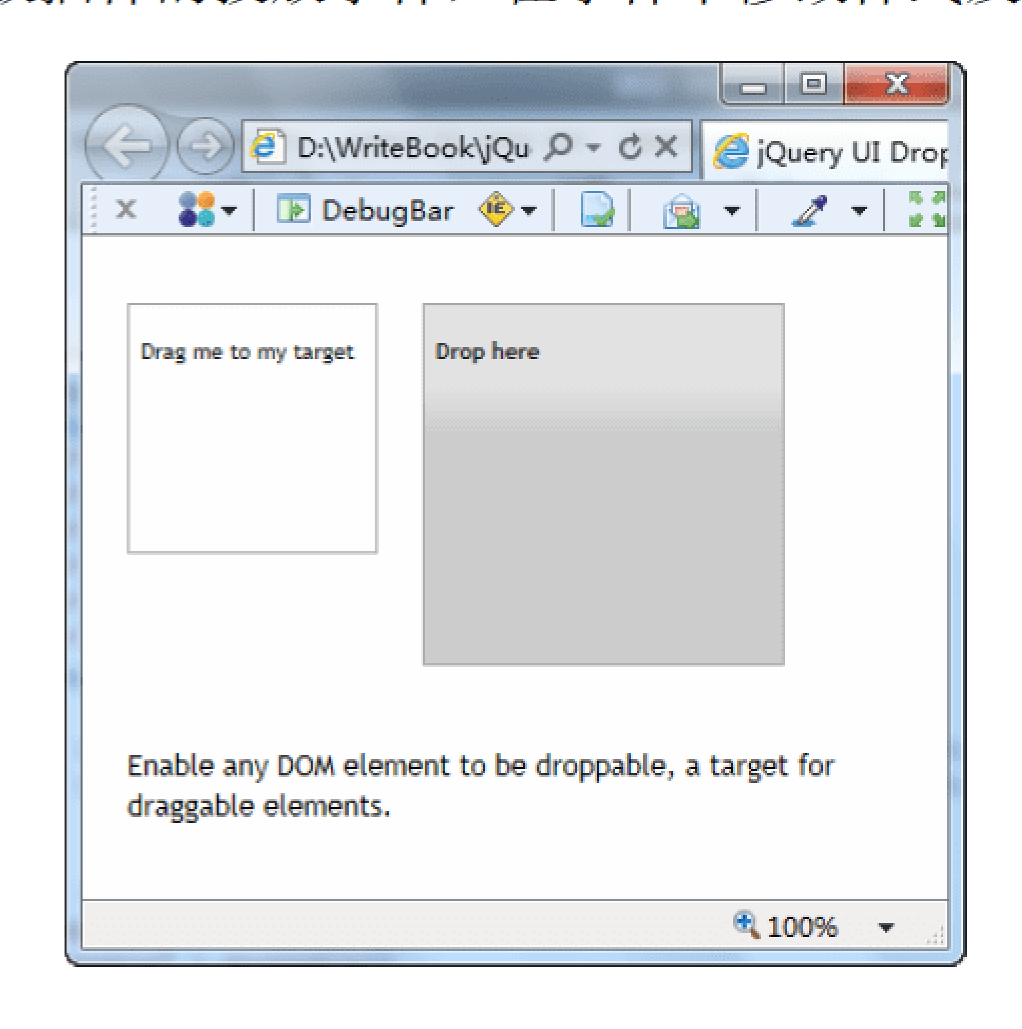


图 14.24　基本投放插件使用

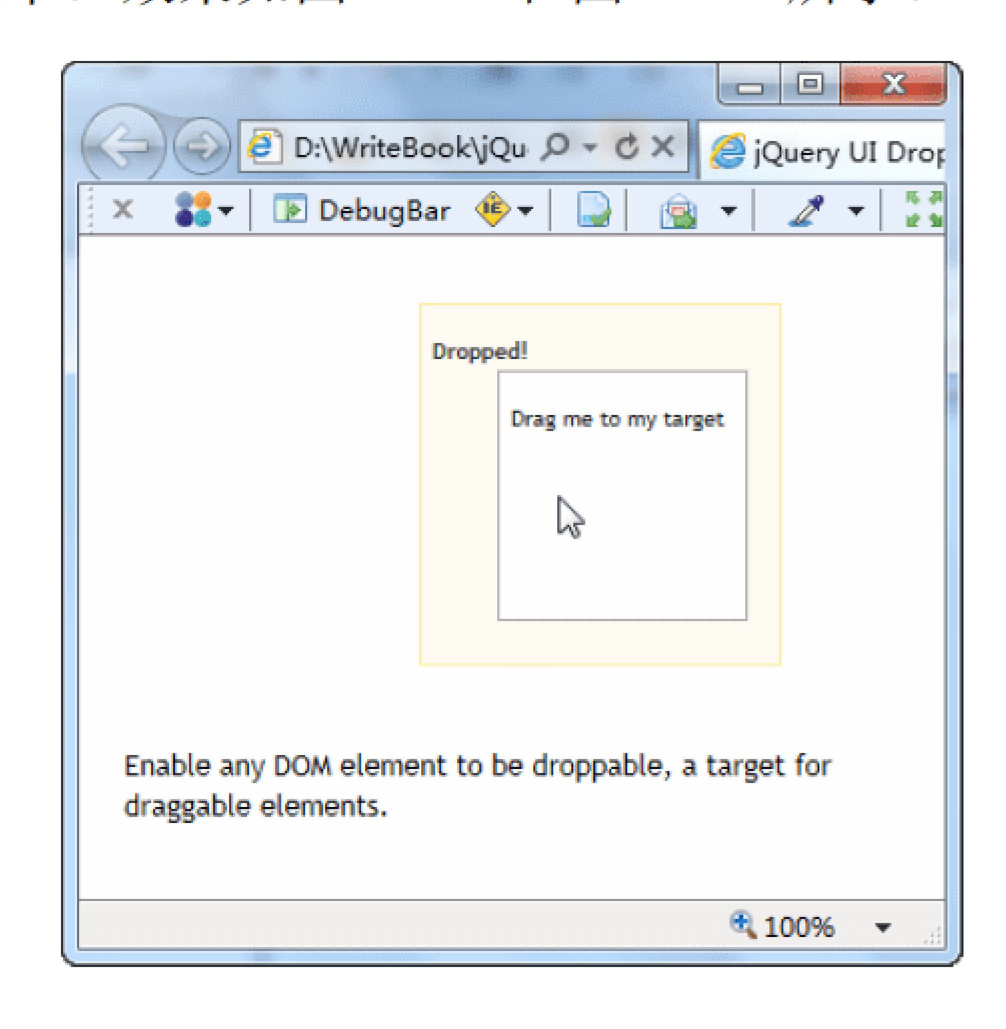


图 14.25　投放插件投放效果

2. 接受拖动元素测试

这个示例中使用了两个拖动元素、一个投放元素。在投放元素中只接受其中一个拖动元素。这里使用了投放元素的 accept 属性指定可接受的拖动元素对象，并使用了 activeClass 属性设定当投放元素被激活时的样式。使用 hoverClass 属性设定当有元素覆盖到投放元素上时的样式，同时也使用了投放事件 drop 修改投放元素样式及文本。

```
1   <script>
2   $(function() {
3       $( "#draggable, #draggable-nonvalid" ).draggable();
4       $( "#droppable" ).droppable({
5           accept: "#draggable",               //接受元素指定
6           activeClass: "ui-state-hover",      //被激活时样式
7           hoverClass: "ui-state-active",      //有其他元素悬停其上时的样式
8           drop: function( event, ui ) {
```

```
9                $( this )
10                   .addClass( "ui-state-highlight" )
11                   .find( "p" )
12                       .html( "Dropped!" );
13           }
14       });
15   });
16   </script>
```

上述代码第 3 行创建了两个拖放元素。第 5 行设定了投放元素可以接受的拖动元素。第 6 行设定投放元素激活样式。第 7 行设定当可接受的拖动元素悬停在投放元素上时的样式。第 8～13 行响应投放插件的投放事件，在事件中修改样式及文本。效果如图 14.26～图 14.28 所示。

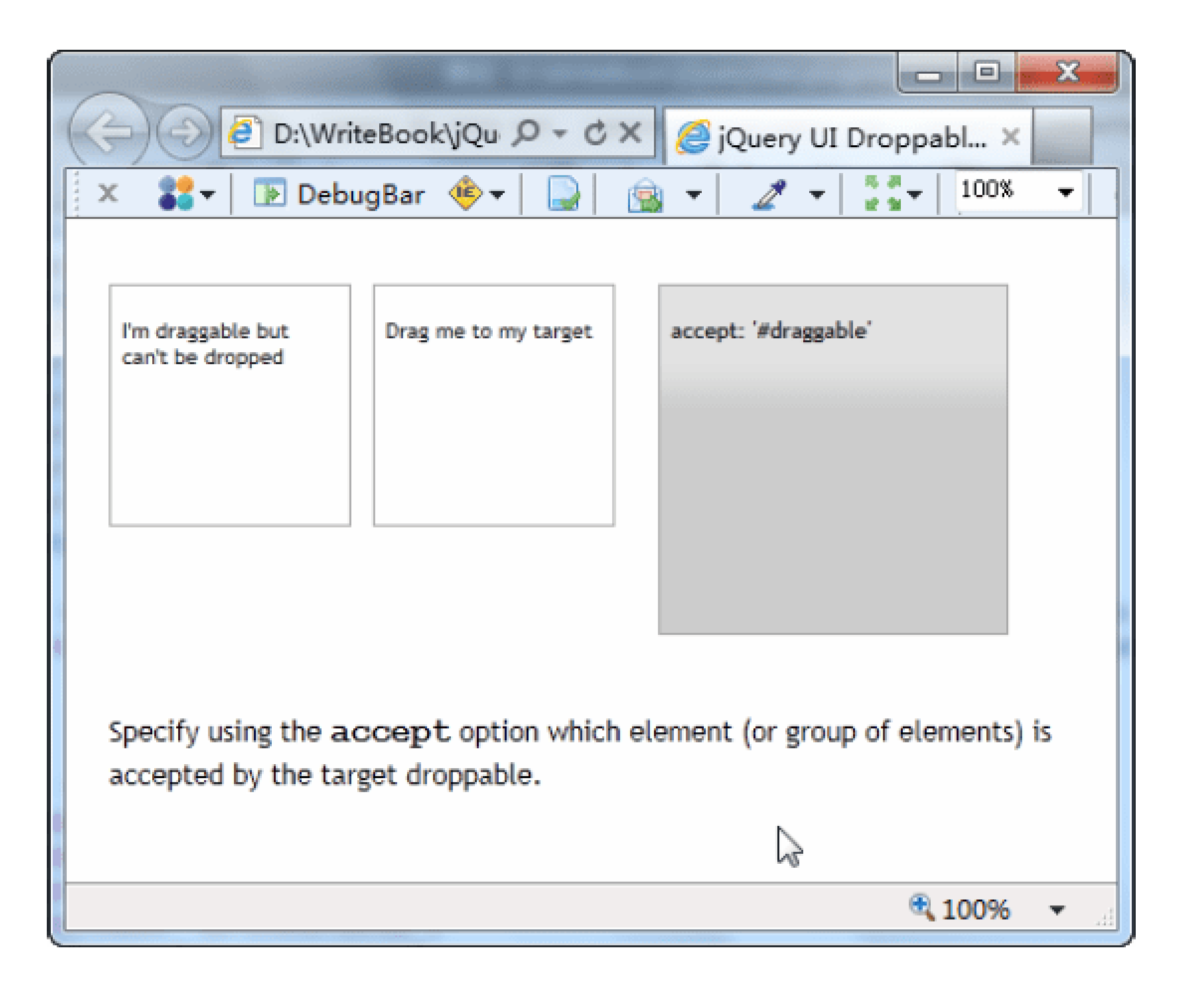


图 14.26　选择拖动元素

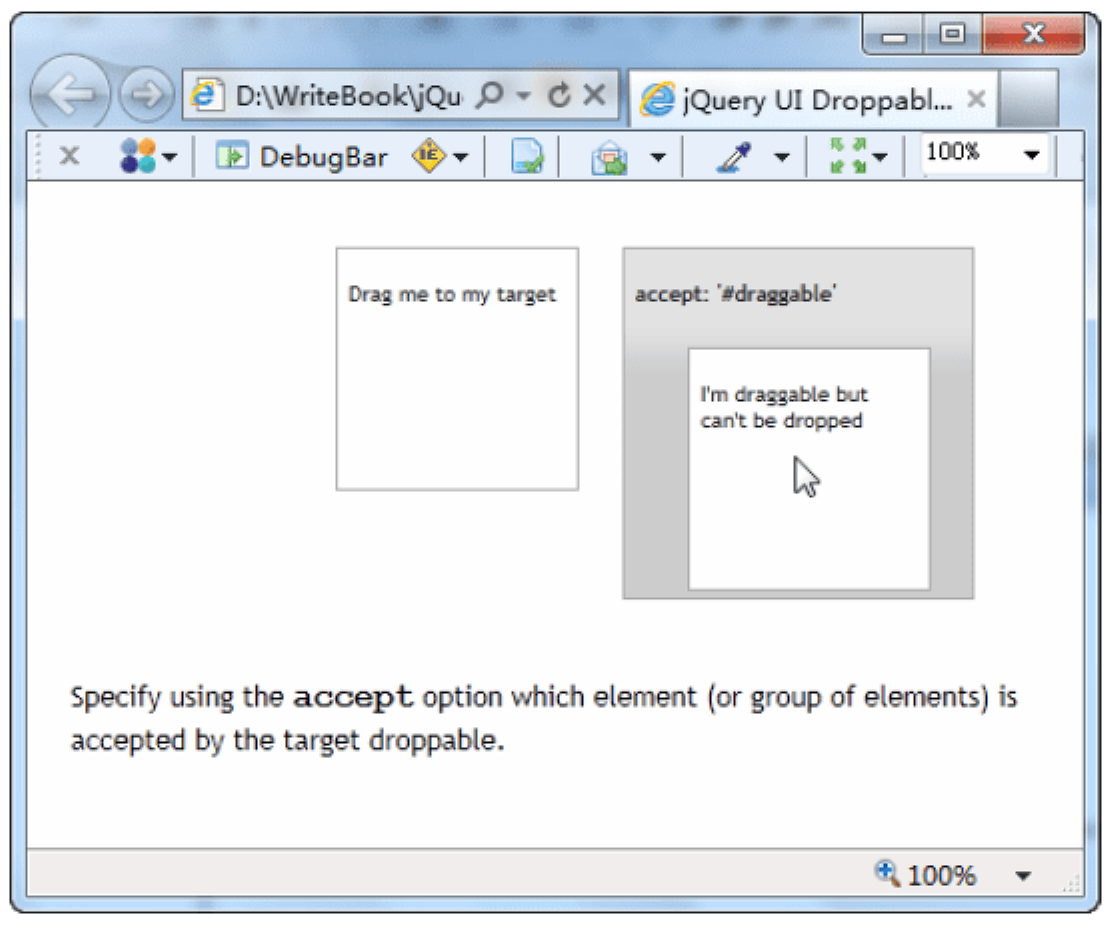


图 14.27　不被接受的拖动元素

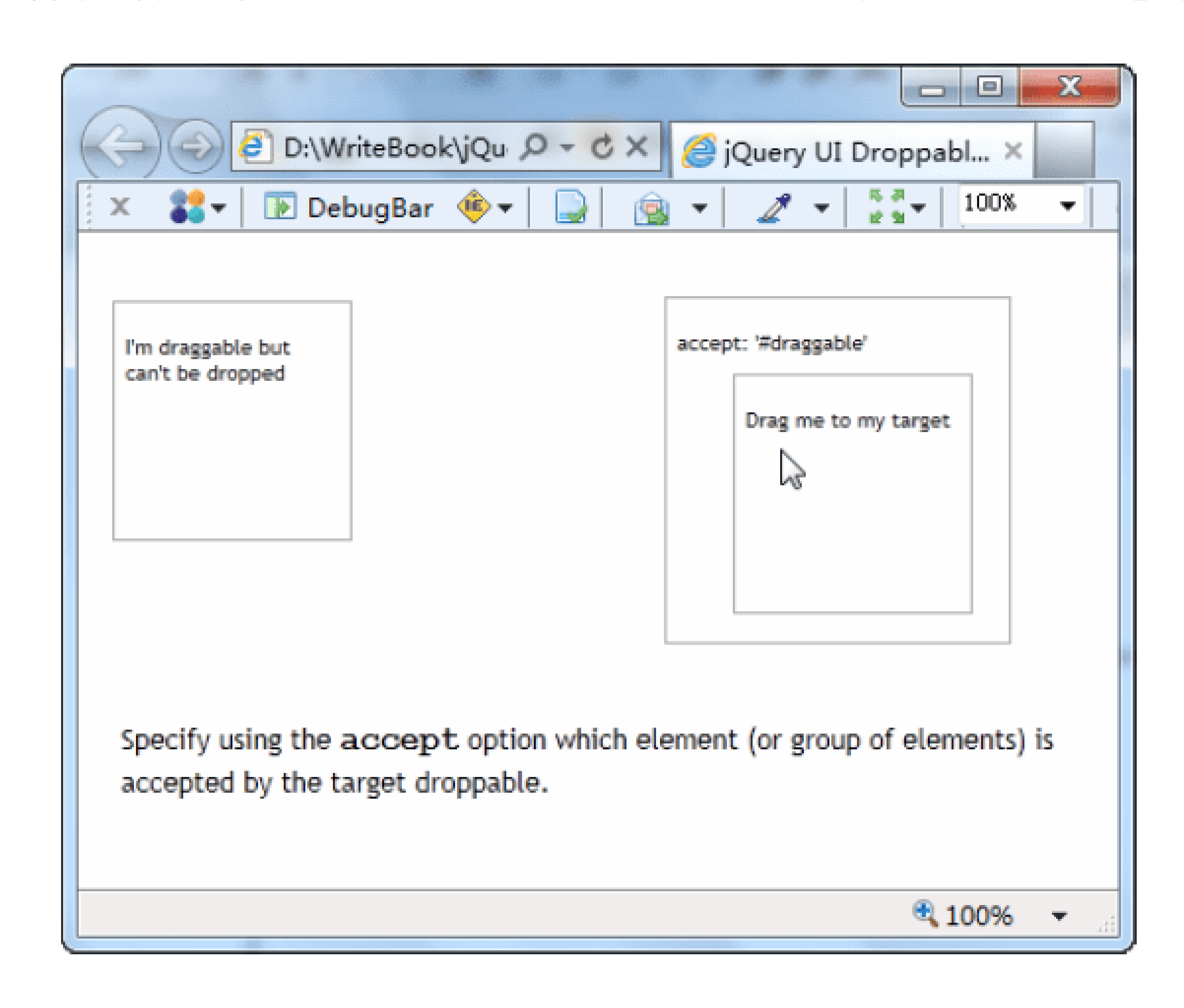


图 14.28　被接受的拖动元素

3．限定事件传递

这个示例使用了两组投放元素、一个拖动元素，一组不限定事件传递，另一组限定事

件传递。当在 drop 事件中设定返回值为 false 时，上层（小）的投放元素的投放事件会传递到下层（大）的投放元素中，如果不设定返回值为 false，则下层（大）投放元素不会产生投放事件。

```
1   <script>
2   $(function() {
3       $( "#draggable" ).draggable();
4       $( "#droppable, #droppable-inner" ).droppable({
5           activeClass: "ui-state-hover",
6           hoverClass: "ui-state-active",
7           drop: function( event, ui ) {
8               $( this )
9                   .addClass( "ui-state-highlight" )
10                  .find( "> p" )
11                      .html( "Dropped!" );
12              return false;
13          }
14      });
15      $( "#droppable2, #droppable2-inner" ).droppable({
16          greedy: true,
17          activeClass: "ui-state-hover",
18          hoverClass: "ui-state-active",
19          drop: function( event, ui ) {
20              $( this )
21                  .addClass( "ui-state-highlight" )
22                  .find( "> p" )
23                      .html( "Dropped!" );
24          }
25      });
26  });
27  </script>
```

上述代码第 16 行组织了事件传递效果。其他代码和前面的例子类似。效果如图 14.29～图 14.31 所示。

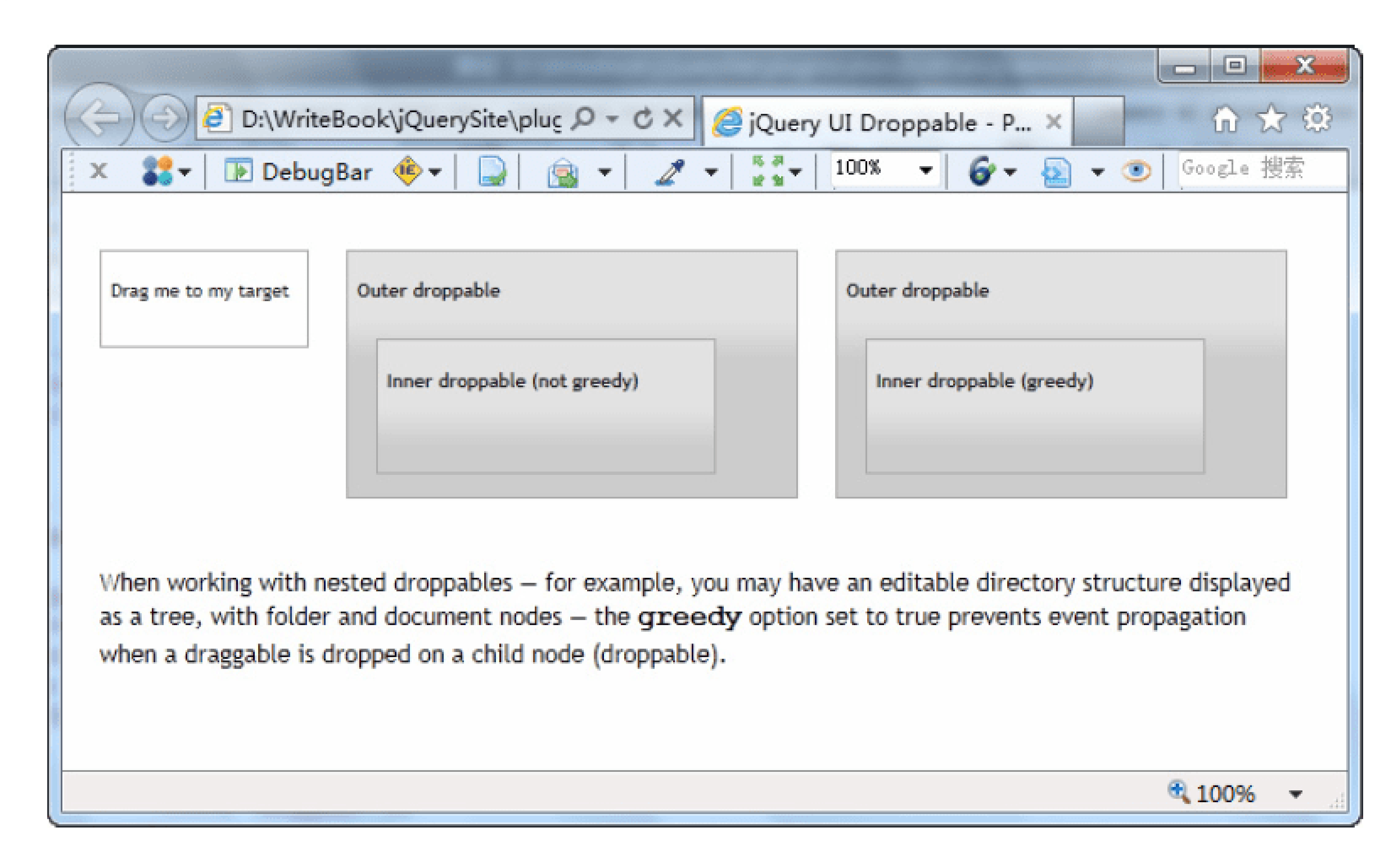


图 14.29　限定事件传递

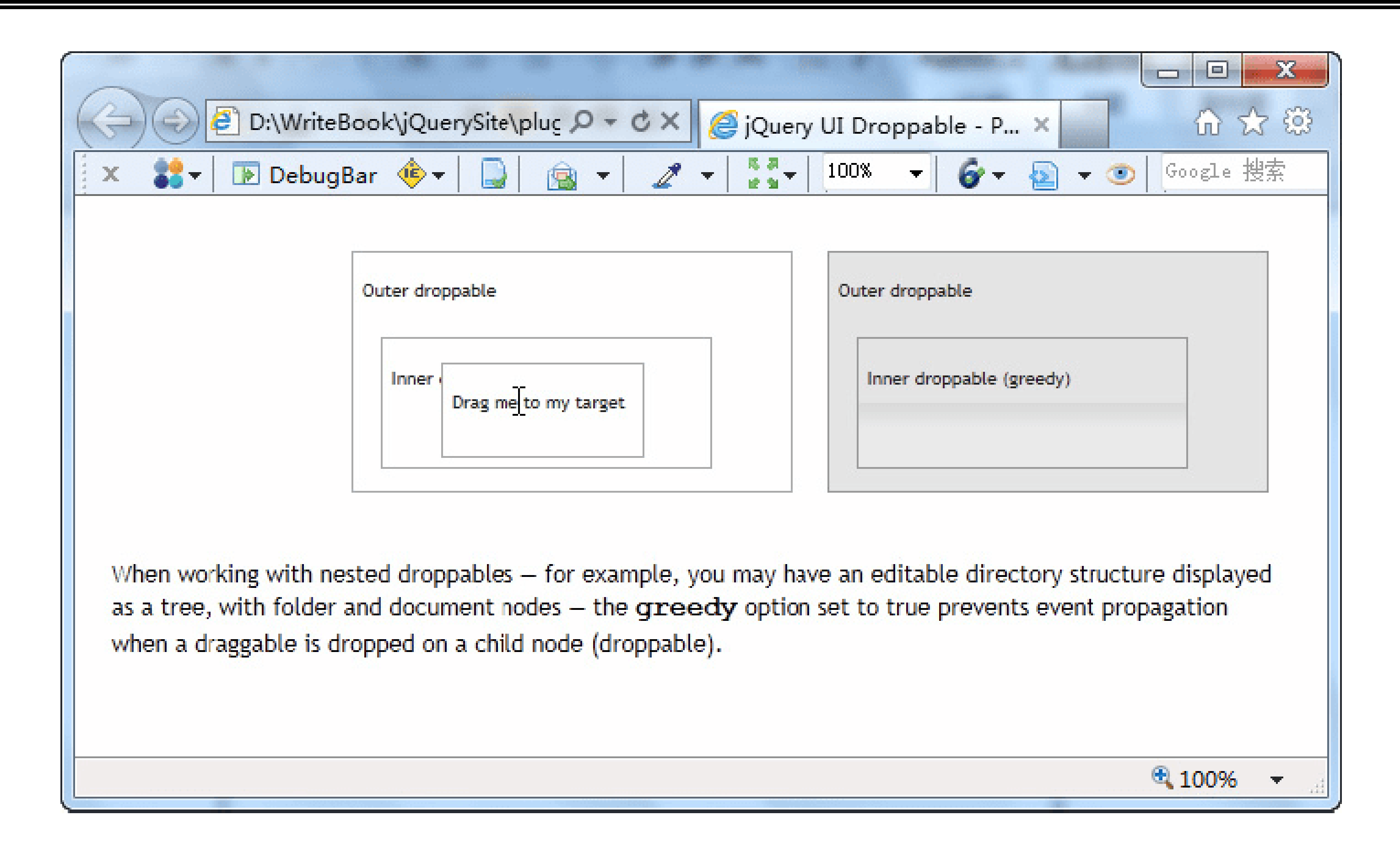


图 14.30　未阻止事件传递

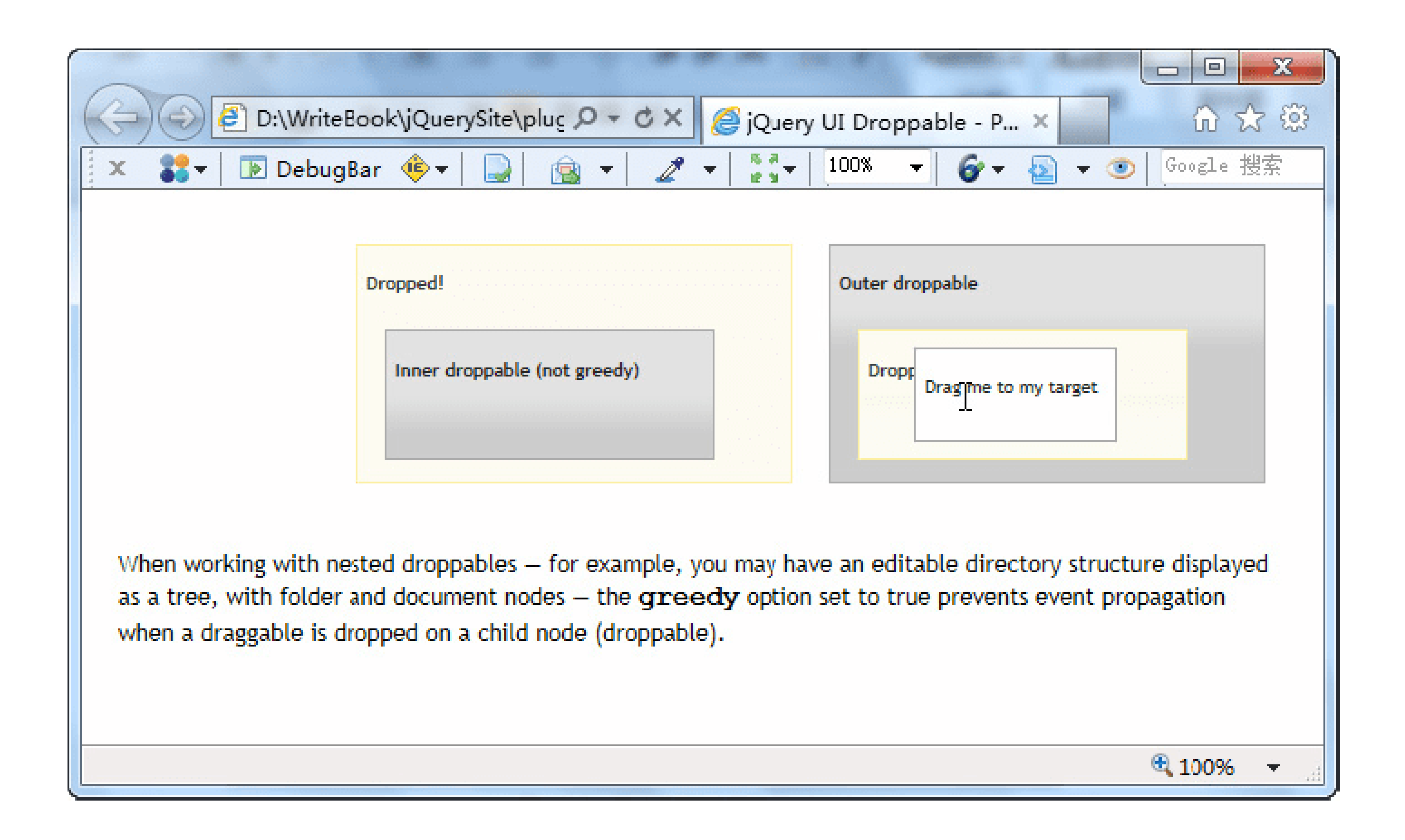


图 14.31　阻止了事件传递

4．购物车效果

这个示例使用了折叠插件、拖动插件、投放插件和排序表格插件，实现拖放操作购物车效果。

```
1   <script>
2   $(function() {
3       $( "#catalog" ).accordion();
4       $( "#catalog li" ).draggable({
5           appendTo: "body",        //拖动元素添加到哪一个元素中
6           helper: "clone"          //设定克隆行为
7       });
8       $( "#cart ol" ).droppable({
9           activeClass: "ui-state-default",
10          hoverClass: "ui-state-hover",
11          accept: ":not(.ui-sortable-helper)",   //接受元素时样式
12          drop: function( event, ui ) {
13              $( this ).find( ".placeholder" ).remove();
14              $( "<li></li>" ).text( ui.draggable.text() ).appendTo( this );
15          }
```

```
16      }).sortable({                                //可排序表格
17          items: "li:not(.placeholder)",           //表格
18          sort: function() {
19              $( this ).removeClass( "ui-state-default" );
20          }
21      });
22  });
23  </script>
```

上述代码第 3 行对 ID 为 catalog 的层创建折叠元素。第 4～7 行创建了多个拖动元素，这些元素添加到 body 中，每个元素在拖动时都被克隆。第 8～15 行创建投放元素。第 9 行设定投放元素激活样式。第 10 行设定当可接受的拖动元素悬停在投放元素上时的样式。第 11 行设定投放元素可接受的拖动元素为去除了样式为.ui-sortable-helper 的元素。第 13 行删除投放元素中的提示内容。

第 14 行向投放元素中加入一个列表项，列表项的文本是鼠标所拖动的元素的文本。第 16～23 行在投放元素的基础上初始化一个排序列表。第 17 行指定排序元素除了投放元素中提示信息以外的所有列表项。第 18 行设定排序功能函数。第 19 行删除样式类 ui-state-default。效果如图 14.32 所示。

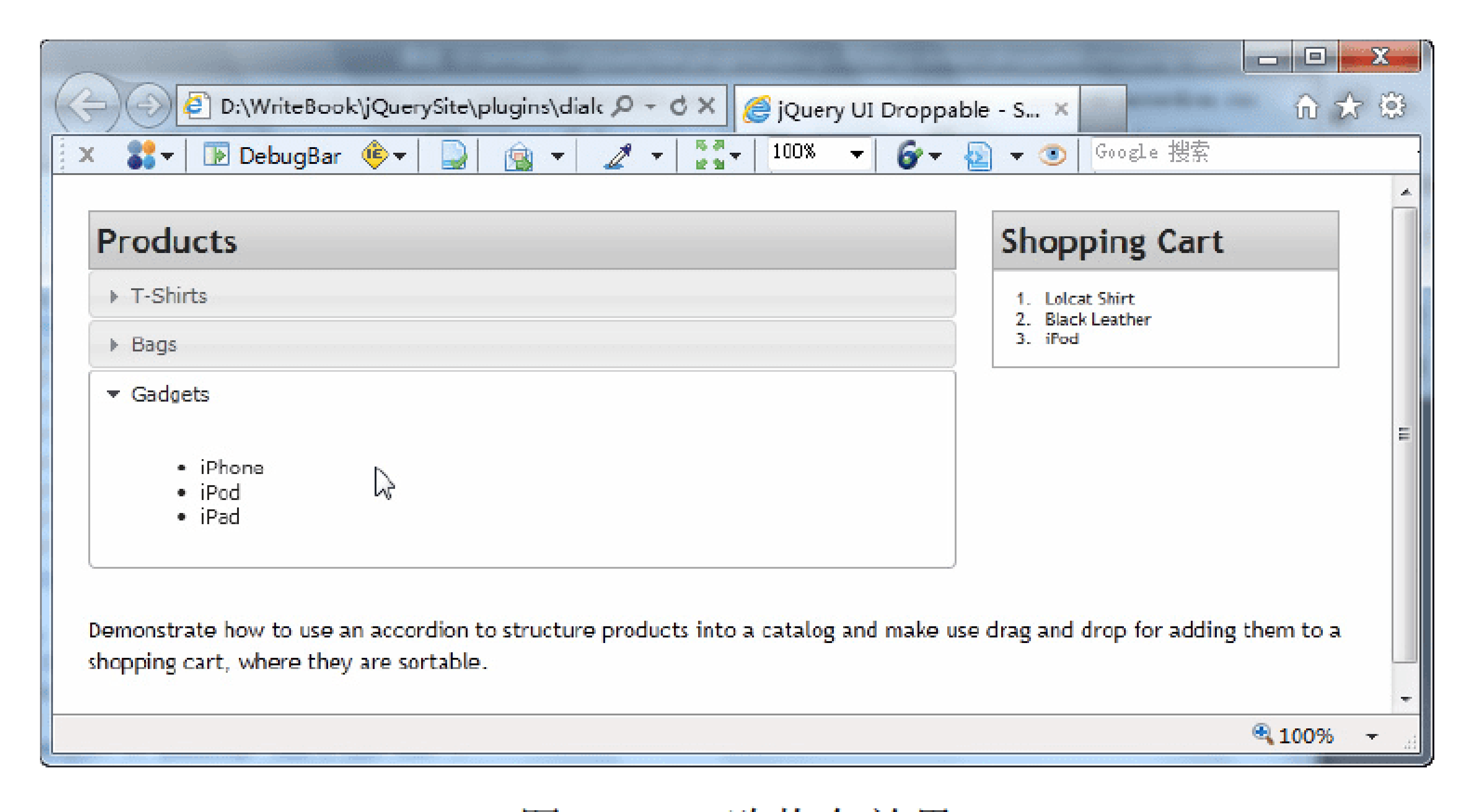


图 14.32　购物车效果

14.3　小　　结

网页元素拖放效果是现代网页中常用到的一种效果。本章主要介绍了 jQuery 拖动插件的基础知识，以及如何利用拖动插件。重点内容是如何利用拖动插件，难点是拖动插件的配合使用。下一章将介绍 jQuery 自定义插件。

14.4　习　　题

【习题 1】练习 jQuery UI 中的拖动插件使用方法。

【习题 2】练习 jQuery UI 中的投放插件使用方法。

第 15 章　插 件 开 发

前面的章节除了讲解如何用 jQuery 实现特效，还讲解了很多 jQuery 插件。jQuery 插件是代码重用的一个很好的体现。那么如何开发 jQuery 插件？开发插件过程中需要用到哪些知识？需要注意什么？这正是本章要讲解的主要内容。

15.1　jQuery 插件开发基础

本节介绍在 jQuery 开发过程中需要用到哪些基础知识，插件与函数之间的区别是什么。

15.1.1　jQuery 插件介绍

jQuery 插件总体来说是对 jQuery 框架的功能补充，有着相对独立的功能，并且自成体系。它有着独立的文件和方法，通过接口进行调用。这里所说的接口就是前面在介绍插件时使用的初始化函数及其他功能函数。更形象地说，jQuery 插件是提供各种各样功能扩展的 jQuery 模块，类似于计算机的各种功能扩展卡。

1．jQuery插件分类

这里所说的分类不是指功能分类，而是在开发 jQuery 插件的过程中，开发形式上的分类。

- 封装对象方法的插件：这种插件也叫对象级别插件，它主要为对象添加方法，并将这些方法封装到对象中。
- 封装全局函数的插件：这种插件也叫类级别插件，类级别插件的开发最直接的理解就是给 jQuery 类添加类方法，可以理解为添加静态方法。典型的示例就是$.AJAX()这个函数，将函数定义于 jQuery 的命名空间中。
- 扩展选择器插件：这种插件是在原有 jQuery 选择器的基础上对特定对象扩展使用特殊选择器。例如，MoreSelectors For Jquery 就是一个典型的 jQuery 选择器扩展。

2．jQuery插件与jQuery固有函数的区别

jQuery 固有函数是 jQuery 框架内置的函数，不需要添加额外文件引用。而 jQuery 插件则是在 jQuery 框架基础上建立起来的功能模块。在插件内同样需要使用 jQuery 内置函数。插件是需要 jQuery 固有工具函数支持的。

15.1.2　jQuery 插件开发基础知识

开发 jQuery 插件需要掌握如下知识点。

（1）HTML/XHTML，这个不用多说，我们的 jQuery 的所有实现效果最后都要通过标记来实现。

（2）CSS，在前面使用插件的过程中我们发现，很多插件除了有自己的功能代码文件外，还有配套的 CSS 文件，这些文件主要在插件动态改变元素样式时使用。

（3）JavaScript，jQuery 框架本身就是由 JavaScript 编写的，我们在使用 jQuery 时也是通过 JavaScript 语言来操作的。

jQuery 框架中需要注意的内容如下。

（1）选择器，$是 jQuery 的简写形式。所以，jQuery()和$()的意思是一样的。所有用$()的地方，$都可以用 jQuery 代替。

（2）插件类型声明，用 jQuery.extend 增加的函数，或者说扩展的函数，可以理解成添加类方法，用类名调用；用 jQuery.fn.extend 增加的函数，或者说扩展的函数，可以理解成添加对象方法，即添加成员函数，用对象名调用。

（3）对象原型，jQuery.fn=jQuery.prototype，所以，jQuery.fn 是 jQuery.prototype 的别名。

（4）内置工具函数。

15.1.3　创建一个简单 jQuery 插件

下面通过一个示例来初步认识一下 jQuery 插件开发过程。这个示例是封装对象方法的插件示例。

首先，声明插件，定义一个新的函数原型到 jQuery.fn 对象中，函数原型的名字就是插件的名字：

```
jQuery.fn.myPlugin = function() {
  //添加插件成员
};
```

在上面的代码中定义了一个名为 myPlugin 的插件。上面的写法还是有些问题，因为如果像上面那样写，可能会导致与其他 jQuery 库冲突。为了解决这个问题，可以将上面的写法修改为：

```
(function( $ ){
  $.fn.myPlugin = function() {
    // 添加插件成员
  };
})( jQuery );
```

上面这种写法叫封闭或者闭包，在封闭的函数内部可以使用 jQuery 特有的$符，而不会出现任何问题。还有一点要说明的就是在插件内部代码中出现的 this 关键字，它代表调

用当前编写的插件的 jQuery 对象，因此我们不用再次在插件内包装 this 关键字。

1. this关键字的使用

下面通过示例代码来使用 this 关键字。

```
1 (function( $ ){
2  $.fn.myPlugin = function() {
    //在代码中不需要写$(this)，因为 this 本身就代表 DOM 元素对象
3   this.fadeIn('normal', function(){
4     // 功能代码
5   });
6  };
7  })( jQuery );
```

调用插件代码可以这样写：

```
$('#element').myPlugin();
```

下面补充代码：

```
1(function( $ ){
2  $.fn.maxHeight = function() {
3     var max = 0;
4   this.each(function() {
5     max = Math.max( max, $(this).height() );
6   });
7   return max;
8  };
9})( jQuery );
```

这个插件完成的功能很简单，实现对所有匹配元素的高度取最大值。可以使用一个 HTML 文件来进行插件测试。首先，在页面建立 4 个<DIV>，分别设置不同内容：

```
1 <div id="div1">jQuery<br />jQuery</div>
2 <div id="div2">jQuery<br />jQuery<br />jQuery<br />jQuery</div>
3 <div id="div3">jQuery<br />jQuery<br />jQuery<br />jQuery<br />jQuery<br />
  jQuery</div>
4 <div id="div4">jQuery<br />jQuery<br />jQuery<br />jQuery<br />jQuery<br />
  jQuery<br />jQuery<br />jQuery</div>
```

在 CSS 样式设定中分别对它们使用不同的背景颜色，代码参考光盘内容。调用插件形式如下：

```
1 <script src="jslib/jquery-1.6.js" type="text/javascript"></script>
2 <script src="js/MyPlugin.js" type="text/javascript"></script>
3 <script type="text/javascript">
4   $(function(){
5       alert("各层最大高度为: "+$("div").maxHeight()+"px");
6   });
7 </script>
```

效果如图 15.1 所示。

2. 插件中this对象的返回

刚才编写的插件很简单，而且返回的内容也只是一个整型值。但是，对于 jQuery 来说它的优势之一是函数链，也就是一个对象可以一次性通过调用一系列的函数完成某个功能。这需要 jQuery 函数能如实返回调用它的对象。因此，为了保证 jQuery 的链接特性，我们要保证编写的插件能够返回 this 对象。下面在原有插件的基础上进行修改，完成返回 this 对象功能。

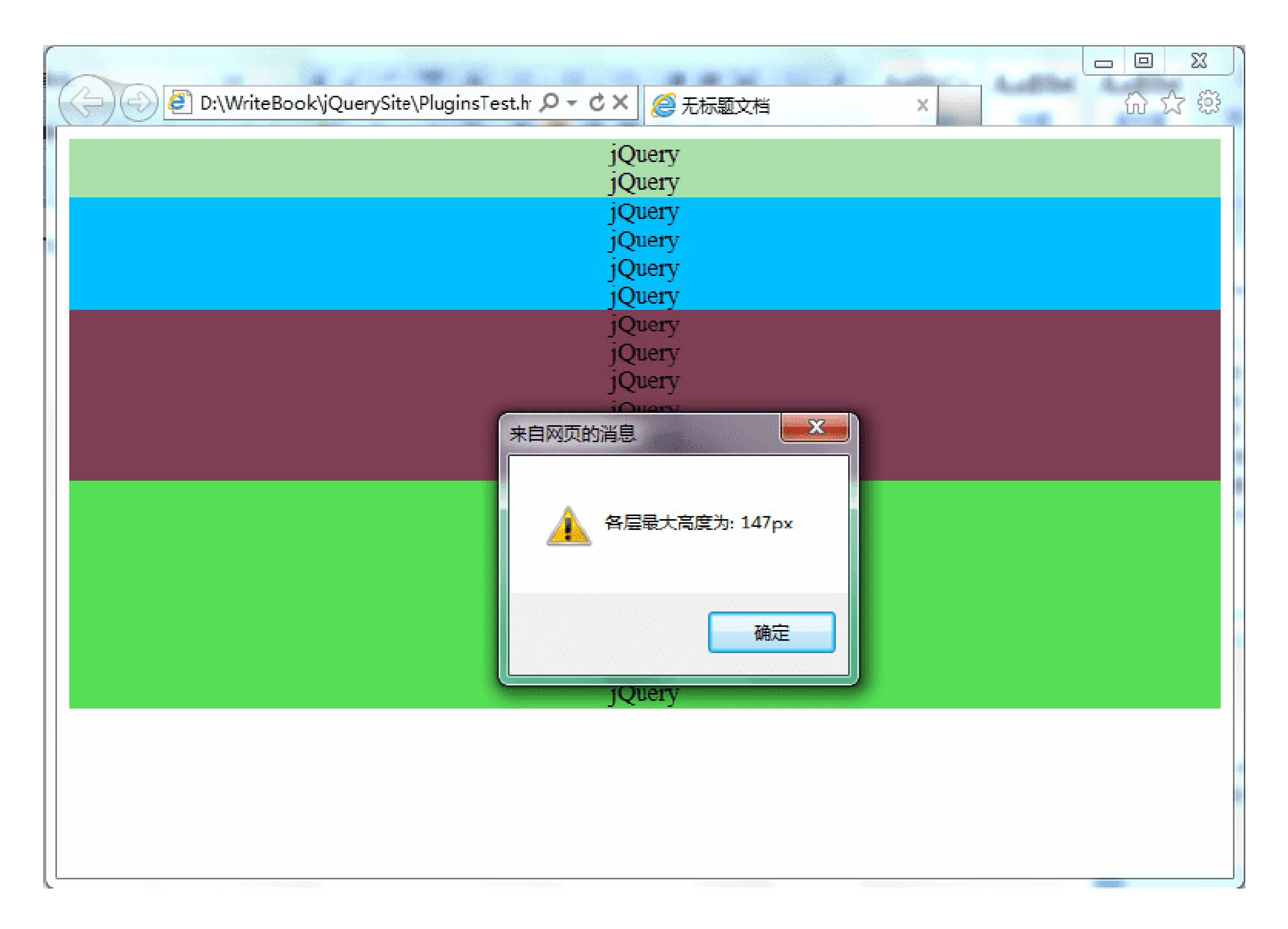


图 15.1　自定义插件示例一

```
1 (function( $ ){
2  $.fn.lockDimensions = function( type ) {
3    return this.each(function() {
4      var $this = $(this);
5      if ( !type || type == 'width' ) {
6        alert($this.width());
7      }
8      if ( !type || type == 'height' ) {
9        alert($this.height() );
10       }
11    });
12  };
13})( jQuery );
```

可以在页面上测试这个插件的作用，以及它是否可以完成可链接操作。利用页面元素调用插件函数名（插件函数名即我们在插件中定义的$.fn.lockDimensions），并根据插件定义定义形式传入参数。

```
1 <script type="text/javascript">
2   $(function(){
3       $("div").lockDimensions('height').css('color', 'red');
4   });
5 </script>
```

HTML 代码和 CSS 样式与上面的例子相同，只是调用插件的方式有所不同。在插件代码的第 3 行就是我们所强调的返回对象本身，也就是将 this 返回。只有这样，在测试代码的第 3 行才可以在调用插件函数结束后继续调用 css()函数设定文字颜色。效果如图 15.2 和图 15.3 所示。

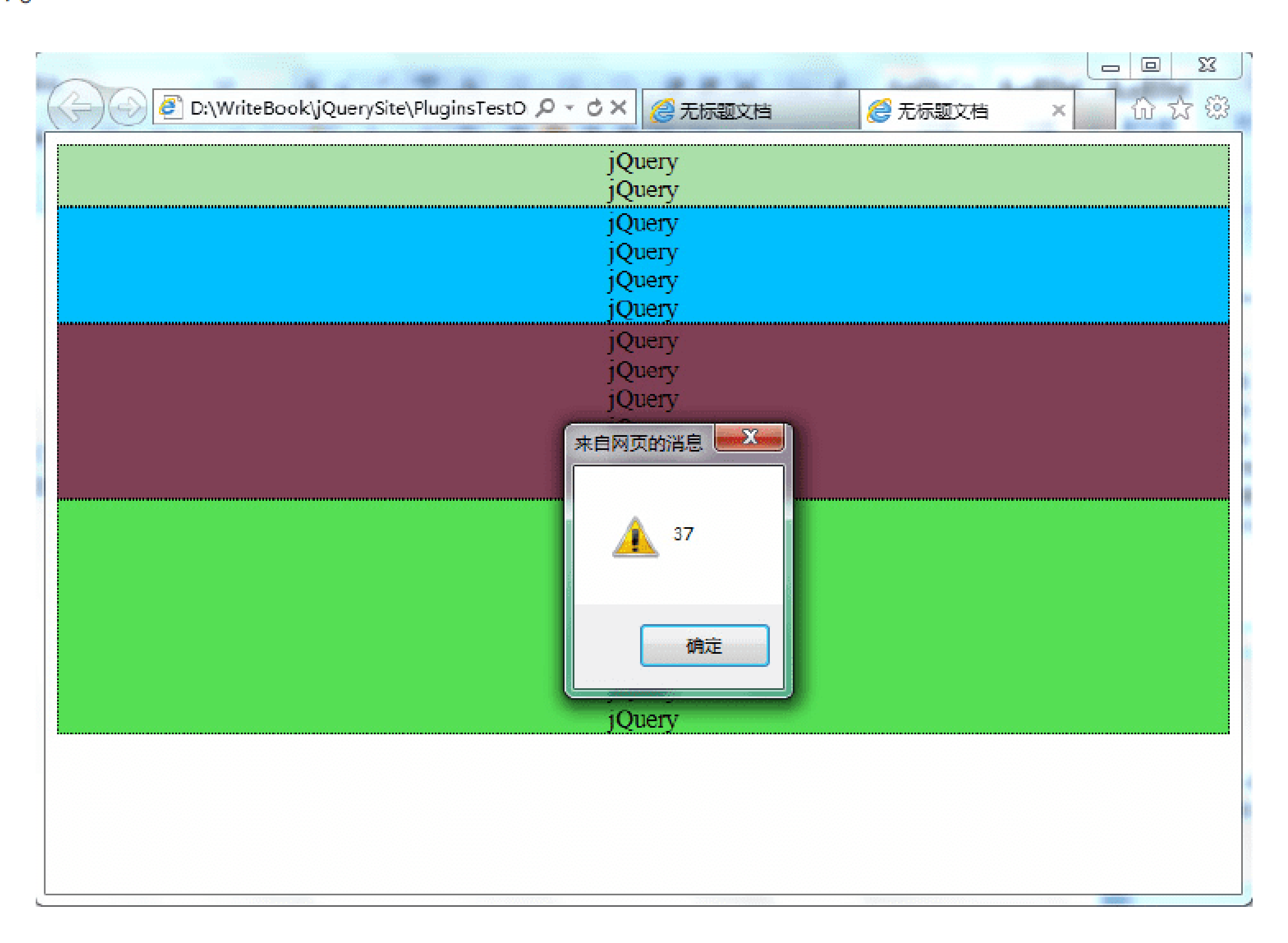


图 15.2　页面加载完成输出各层的高

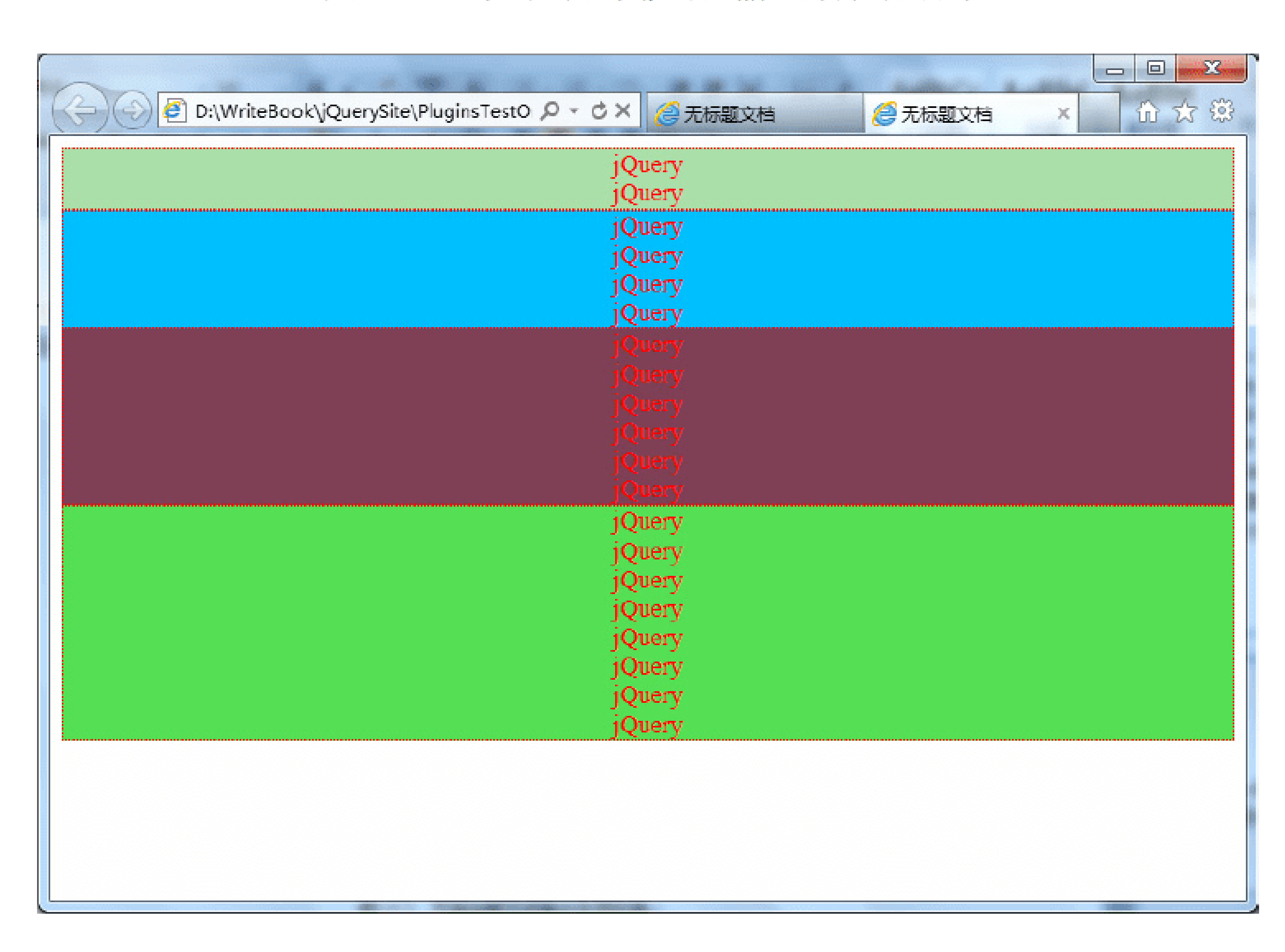


图 15.3　插件功能完成后修改字体颜色

3．以选项对象代替零散参数

自定义插件还要设定默认值与选项。对于较复杂的插件，按照惯例不是采用外部直接传递零散参数形式，而是使用一种可迭代的选项对象来进行功能设定。可以看下面这个插件示例：

```
1 (function( $ ){
2 $.fn.tooltip = function( options ) {
```

```
3    var settings = {
4      'location'          : 'top',
5      'background-color' : 'blue'
6    };
7    return this.each(function() {
8      if ( options ) {
9        $.extend( settings, options );
10       alert("当前传入方向: "+settings["location"]+"; 当前传入背景色: "+settings
         ["background-color"]);
11      }
12    else
13       alert("默认方向: "+settings["location"]+"; 默认背景色: "+settings
         ["background-color"]);
14    });
15  };
16})( jQuery );
```

上述代码第 3～6 行是插件的默认选项设置。第 8 行判断调用插件时是否传入了选项对象。第 9 行将外部传进来的选项对象与默认选项合并。

在调用这个插件的时候，可以给定参数，例如：

```
<script type="text/javascript">
    $(function(){
        $('div').tooltip({
          'location'          : 'bottom',
          'background-color' : 'red'
        });
    });
</script>
```

效果如图 15.4 和图 15.5 所示。

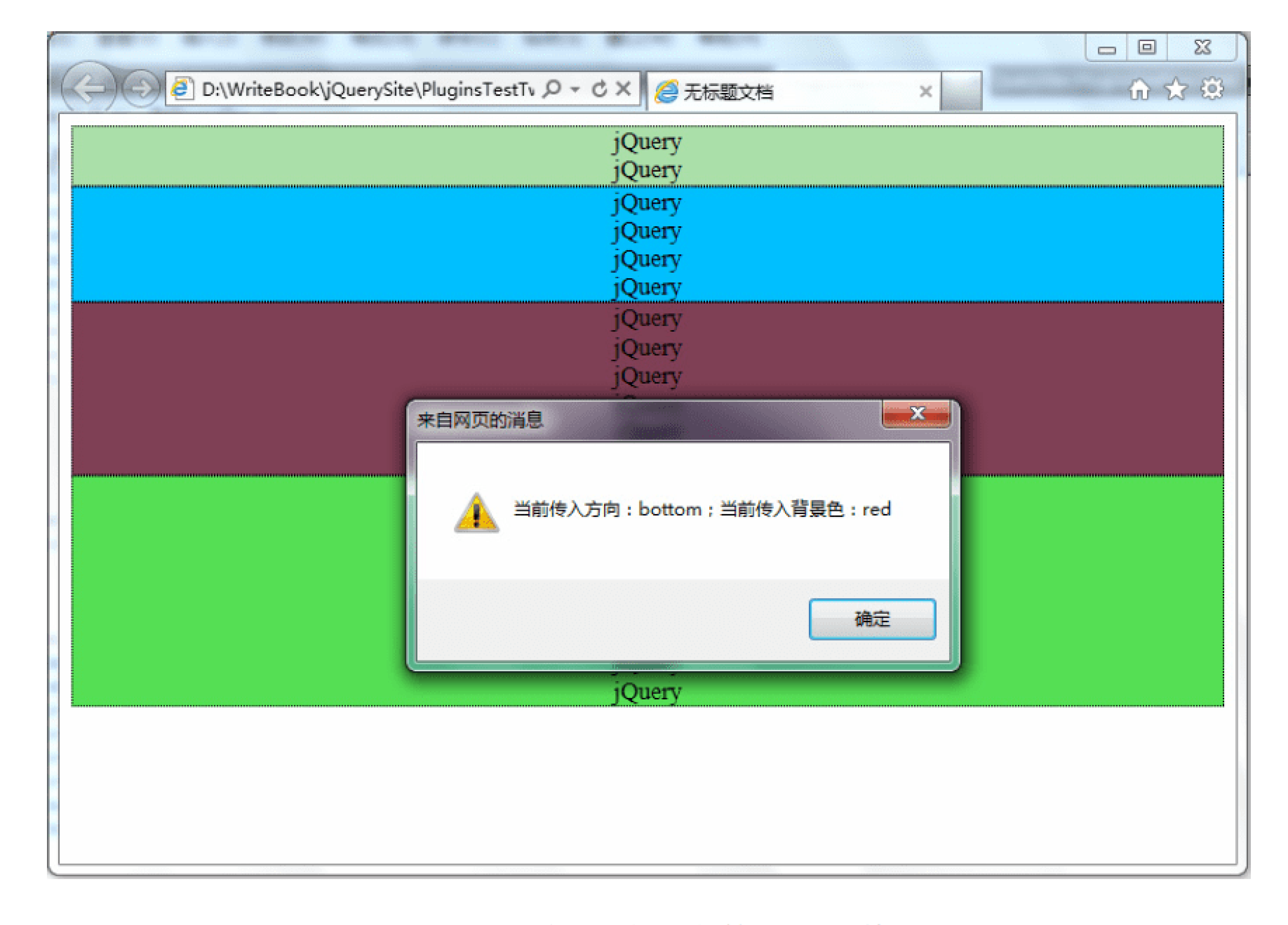


图 15.4　携带选项参数调用插件

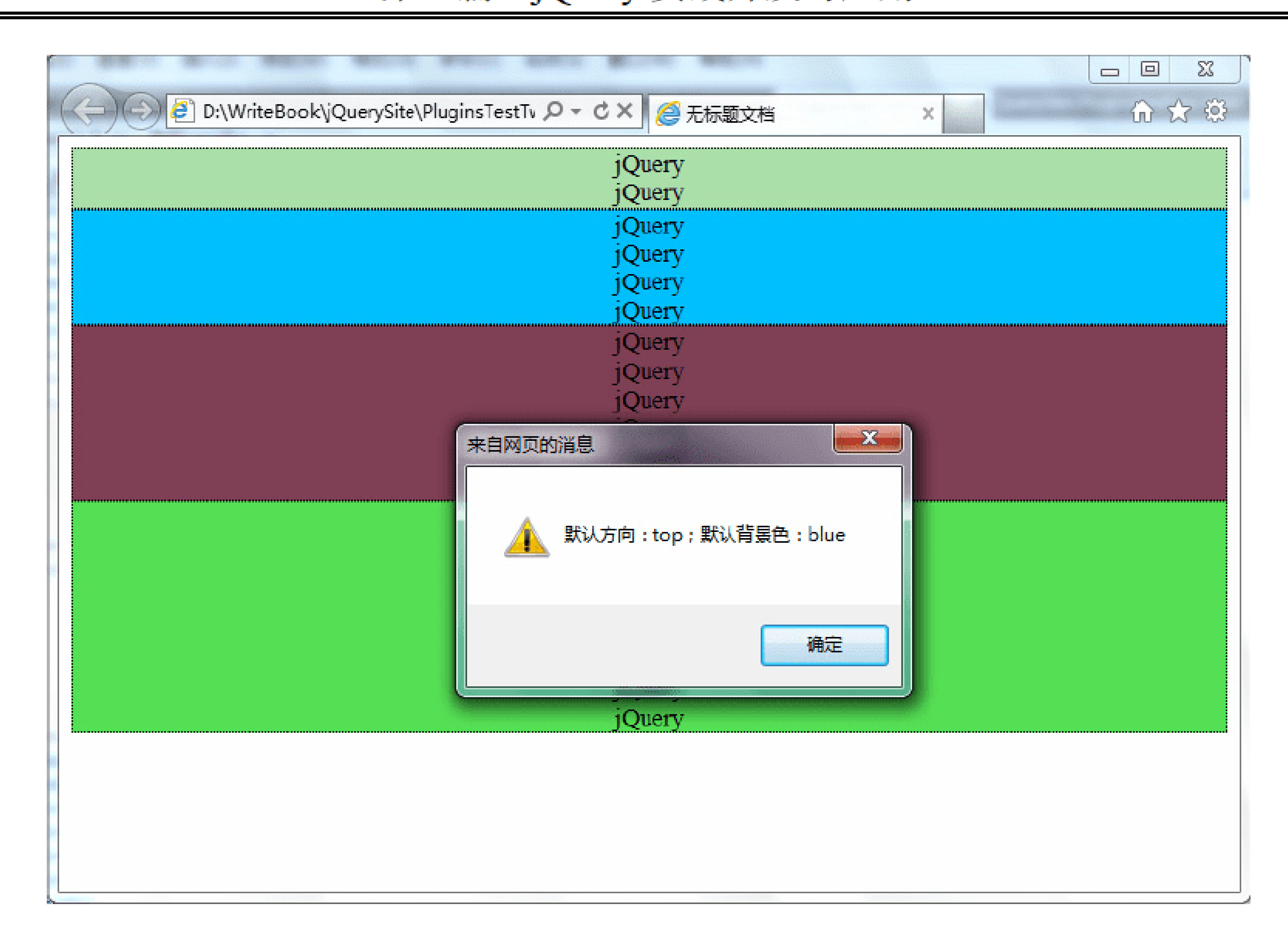


图 15.5　默认选项参数调用插件

4．向插件中添加方法

前面在介绍插件时曾提到插件有一些方法可用。那么这些方法是如何定义在插件中的呢？看下面这个例子：

```
1 (function( $ ){
2  var methods = {
    //声明插件方法
3   init : function( options ) {alert("Initial Method"); },
4   show : function( ) { alert("Show Method");   },
5   hide : function( ) { alert("Hide Method"); },
6   update : function( content ) { alert("Update Method "+"Content:
    "+content); }
7  };
8  $.fn.tooltip = function( method ) {
    //定义方法功能
9   if ( methods[method] ) {
10     return methods[ method ].apply( this, Array.prototype.slice.call
       ( arguments, 1 ));
11   } else if ( typeof method === 'object' || ! method ) {
12     return methods.init.apply( this, arguments );
13   } else {
14     $.error( 'Method ' +  method + ' does not exist on jQuery.tooltip' );
15   }
16  };
17 })( jQuery );
```

上述代码第 3～6 行定义了插件的初始化、显示、隐藏、更新等方法，并加入了简单实现。第 9 行判断调用插件时传递的方法名是否有效。第 10 行调用与传进来的方法名对应的方法执行。第 11 行设定如果方法名为空则为初始化方法。第 13 行表示如果方法名不为空但是不在插件定义的方法范围内，则利用 jQuery 的异常处理抛出异常。

下面对方法的使用进行测试：

```
<script type="text/javascript">
    $(function(){
        $('div').tooltip();
        $('div').tooltip({
          foo : 'bar'
        });
        $('div').tooltip('hide');
        $('div').tooltip('update', 'This is the new tooltip content!');
        $('div').tooltip('show');
    });
</script>
```

效果如图 15.6 和图 15.7 所示。

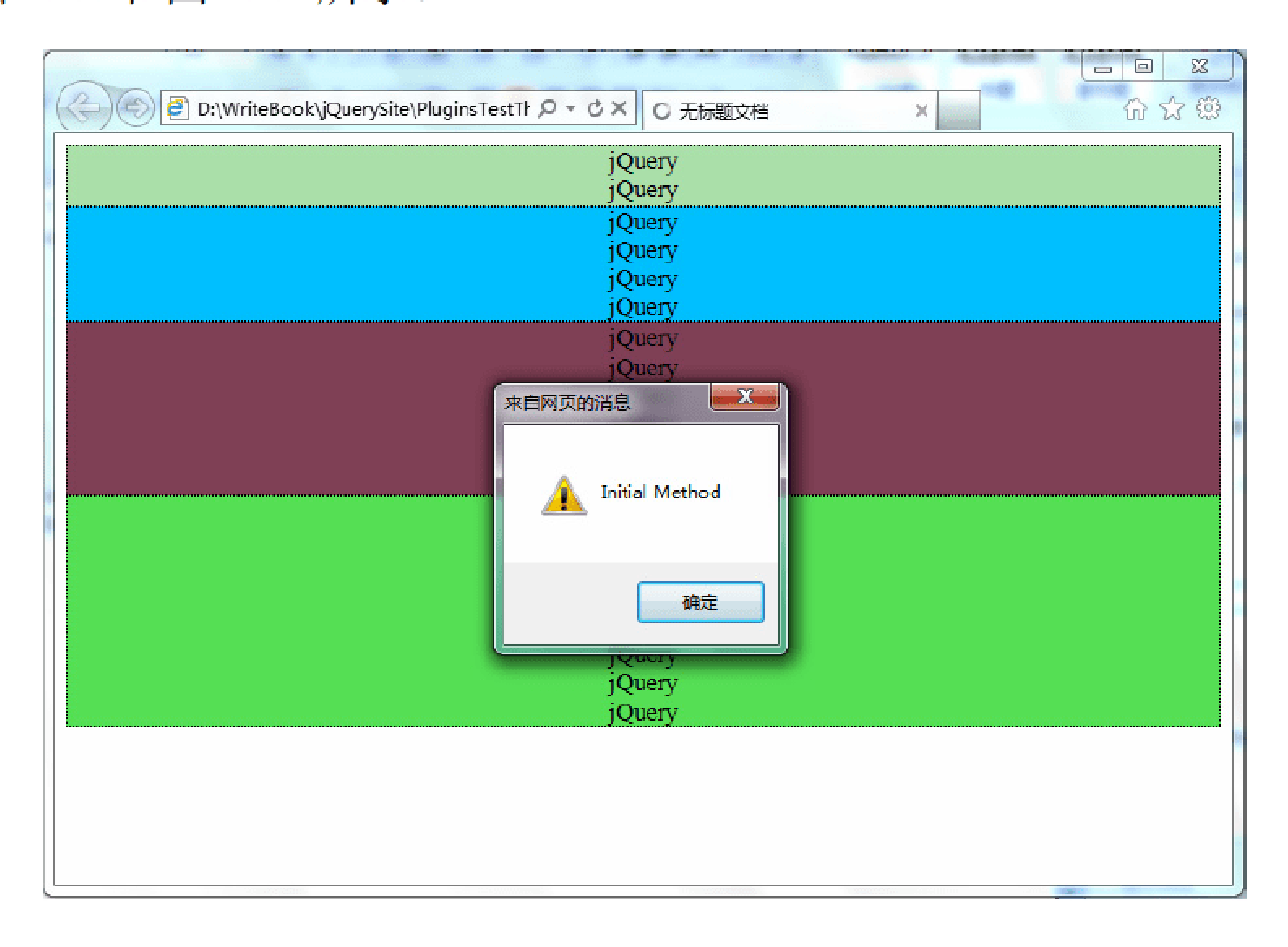


图 15.6　初始化方法调用

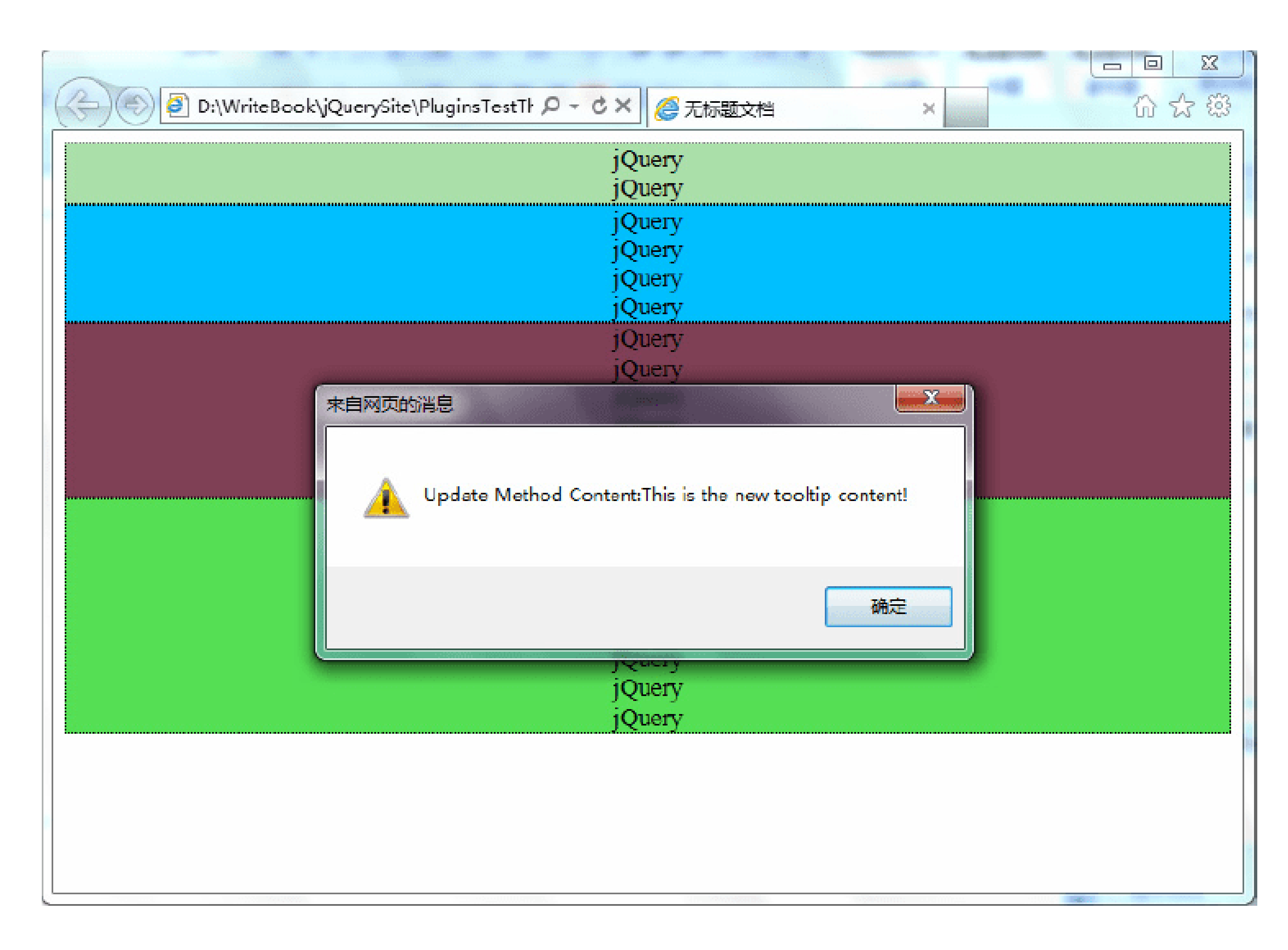


图 15.7　更新方法调用

5．向插件中加入事件

除了可以在插件中声明多个方法之外，还可以对插件绑定事件，使插件能够响应事件并做相应处理。

```
1 (function( $ ){
2  var methods = {
3    init : function( options ) {
4      return this.each(function(){
5        $(window).bind('resize.tooltip', methods.reposition);
6      });
7    },
8    destroy : function( ) {           //声明事件并添加实现
9      return this.each(function(){
10        $(window).unbind('.tooltip');
11      })
12    },
13    reposition : function( ) {alert("窗口当前宽度："+$(window).width()+";
     窗口当前高度："+$(window).height());},
14    show : function( ) {alert("Show Method");},
15    hide : function( ) { alert("Hide Method");},
16    update : function( content ) {alert("Update Method "+"Content:
     "+content); }
17  };
18  $.fn.tooltip = function( method ) {
19    if ( methods[method] ) {
20      return methods[method].apply( this, Array.prototype.slice.call
     ( arguments, 1 ));
21    } else if ( typeof method === 'object' || ! method ) {
22      return methods.init.apply( this, arguments );
23    } else {
24      $.error( 'Method ' +  method + ' does not exist on jQuery.tooltip' );
25    }
26  };
27})( jQuery );
```

上述代码第 5 行在插件初始化时绑定调整大小事件到浏览器窗口，并且指定事件处理程序为函数 reposition()。第 10 行在插件的销毁函数中取消窗口绑定事件。

下面可以测试插件事件：

```
1 <script type="text/javascript">
2  $(function(){
3      $('#div1').tooltip();
4      $('#div1').tooltip({
5        foo : 'bar'
6      });
7      $('#div1').tooltip('hide');
8      $('#div1').tooltip('update', 'This is the new tooltip content!');
9      $('#div1').tooltip('show');
```

```
10      $('#div1').click(function(){$('div').tooltip('destroy')});
11  });
12</script>
```

效果如图 15.8 所示。

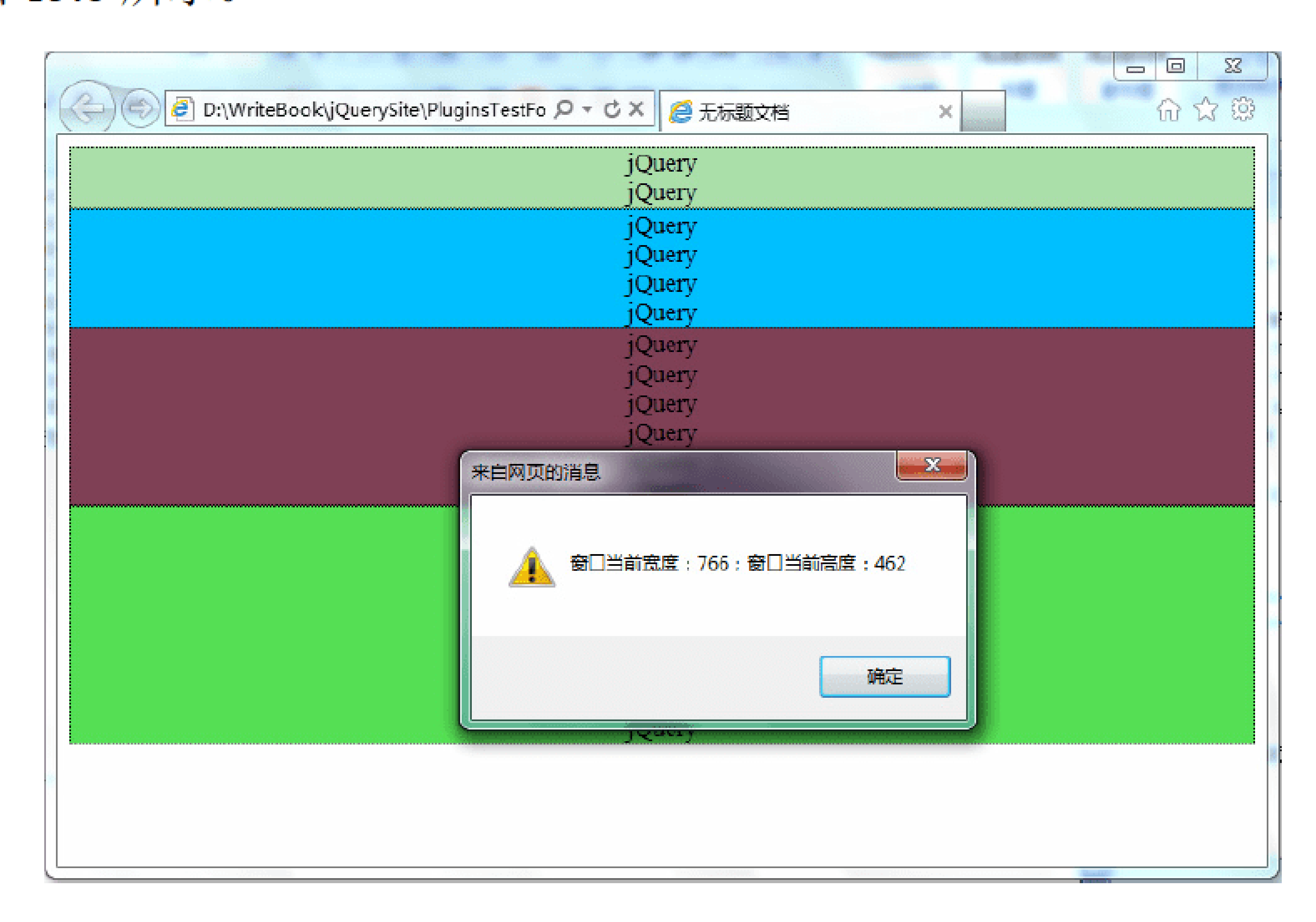


图 15.8　插件事件测试

15.2　插件示例

本节将通过写一个比较具有应用性的插件来完整地体验一下插件开发过程。这个插件所要实现的功能是当鼠标悬停到超链接上时，可以出现背景颜色和文字颜色都不同的提示信息。这里暂且管它叫彩色工具提示插件。

1. 插件原理及显示风格设定

这个插件的原理是利用 HTML 标记中添加的 title 属性，将其转换至页面上，并形成一系列的工具提示。它里面共有 6 种颜色可选，当然，也可以根据个人需要添加更多种颜色选择。这个插件也是一种比较新颖的用户体验。

它的转换机制如下。当在源文件中出现这样一行 HTML 代码时：

```
<a href="index.html" class="blue" title="转到首页">首页</a>
```

使用 jQuery 插件转换后页面应该是下面的样子：

```
<a class="blue colorTipContainer" href="index">首页
<span class="colorTip" style="margin-left: -60px;">回到首页</span>
<span class="pointyTipShadow"></span>
<span class="pointyTip"></span>
</span>
</a>
```

注：上述代码中的 blue 样式类名是为了覆盖默认的提示信息的背景色。

为了能够让工具提示在正确的位置显示，并且显示样式正常，需要定制 CSS 样式文件，这个文件是和插件搭配使用的：

```
1 .colorTipContainer{
2   position:relative;
3   text-decoration:none !important;
4 }
5 .colorTip{
6   /* 彩色工具提示样式类 */
7   display:none;
8   position:absolute;
9   left:50%;
10  top:-30px;
11  padding:6px;
12  background-color:white;
13  font-family:Arial,Helvetica,sans-serif;
14  font-size:11px;
15  font-style:normal;
16  line-height:1;
17  text-decoration:none;
18  text-align:center;
19  text-shadow:0 0 1px white;
20  white-space:nowrap;
21  -moz-border-radius:4px;
22  -webkit-border-radius:4px;
23  border-radius:4px;
24 }
25 .pointyTip,.pointyTipShadow{
26  /* 彩色提示信息下端的三角形样式 */
27  border:6px solid transparent;
28  bottom:-12px;
29  height:0;
30  left:50%;
31  margin-left:-6px;
32  position:absolute;
33  width:0;
34 }
35 .pointyTipShadow{
36  /* 边框阴影样式 */
37  border-width:7px;
38  bottom:-14px;
39  margin-left:-7px;
40 }
```

上述代码中.colorTipContainer 样式类是彩色工具提示的容器元素样式类。为了创建不同颜色的工具提示信息，还需要在 CSS 文件中加入这个插件的相关主题。目前，我们给出了 6 个主题，读者可以根据自己的需要，按照主题格式创建自己的主题风格。

```
/* 6 种主题样式 */

.white .pointyTip{ border-top-color:white;}
```

```
.white .pointyTipShadow{ border-top-color:#ddd;}
.white .colorTip{
    background-color:white;
    border:1px solid #DDDDDD;
    color:#555555;
}

.yellow .pointyTip{ border-top-color:#f9f2ba;}
.yellow .pointyTipShadow{ border-top-color:#e9d315;}
.yellow .colorTip{
    background-color:#f9f2ba;
    border:1px solid #e9d315;
    color:#5b5316;
}

.blue .pointyTip{ border-top-color:#d9f1fb;}
.blue .pointyTipShadow{ border-top-color:#7fcdee;}
.blue .colorTip{
    background-color:#d9f1fb;
    border:1px solid #7fcdee;
    color:#1b475a;
}

.green .pointyTip{ border-top-color:#f2fdf1;}
.green .pointyTipShadow{ border-top-color:#b6e184;}
.green .colorTip{
    background-color:#f2fdf1;
    border:1px solid #b6e184;
    color:#558221;
}

.red .pointyTip{ border-top-color:#bb3b1d;}
.red .pointyTipShadow{ border-top-color:#8f2a0f;}
.red .colorTip{
    background-color:#bb3b1d;
    border:1px solid #8f2a0f;
    color:#fcfcfc;
    text-shadow:none;
}

.black .pointyTip{ border-top-color:#333;}
.black .pointyTipShadow{ border-top-color:#111;}
.black .colorTip{
    background-color:#333;
    border:1px solid #111;
    color:#fcfcfc;
    text-shadow:none;
}
```

至此，显示样式的准备工作基本完成了。下面就可以着手编写插件的 JavaScript 文件了。它的实现步骤如下。

（1）获取调用插件时的主题类名参数。

（2）获取元素的 title 属性，如果元素没有这个属性，则直接停止插件工作。

（3）创建鼠标在元素的悬停与离开事件发生时，插件对定时器及彩色工具提示显示与隐藏的设定。

（4）在元素上创建<span>元素来表示彩色工具提示。

（5）为工具提示选定颜色，如果颜色在主题中被设定，则使用，否则使用默认颜色。

（6）注册产生彩色工具提示的触发事件。

2. 基本插件功能实现

首先，我们来看一下插件的基本功能代码：

```
1  (function($){
2   $.fn.colorTip = function(settings){
3       var defaultSettings = {
4           color       : 'yellow',
5           timeout     : 500
6       }
7       var supportedColors = ['red','green','blue','white','yellow',
        'black'];
8       /*将默认设置合并到设置参数中 */
9       settings = $.extend(defaultSettings,settings);
10      /*
11      *   遍历所有的元素
12      *   将插件添加至方法链中
13      */
14      return this.each(function(){
15          var elem = $(this);
16          //如果当前遍历的元素 title 属性为空则返回
17          if(!elem.attr('title')) return true;
18          //创建事件安排对象及提示信息对象
19          //这两个对象的定义在插件后续部分
20          var scheduleEvent = new eventScheduler();
21          var tip = new Tip(elem.attr('title'));
22          // 添加工具提示到当前遍历的元素上
23          // 并应用插件指定样式
24          elem.append(tip.generate()).addClass('colorTipContainer');
25          // 判断是否提供了插件支持的颜色
26          // 的类名
27          var hasClass = false;
28          for(var i=0;i<supportedColors.length;i++)
29          {
30              if(elem.hasClass(supportedColors[i])){
31                  hasClass = true;
32                  break;
33              }
34          }
35          //如果已经设置了插件支持颜色则使用
36          if(!hasClass){
37              elem.addClass(settings.color);
```

```
38          }
39          //鼠标悬停到元素上则显示提示
40          //鼠标离开则提示延迟隐藏
41          elem.hover(function(){
42              tip.show();
43              //当鼠标移动到元素上
44              //清除先前定义的延迟隐藏设置
45              scheduleEvent.clear();
46          },function(){
47              // 鼠标从元素上离开
48              // 则设定延迟隐藏提示
49              scheduleEvent.set(function(){
50                  tip.hide();
51              },settings.timeout);
52          });
53          // 删除元素上的 title 属性
54          elem.removeAttr('title');
55      });
56  }
57  })(jQuery);
```

上述代码是插件的整体代码，其中第 3～6 行是插件的默认设置。第 7 行设定了插件会支持的颜色选择数组。第 28～34 行判断调用插件所提供的颜色插件是否支持。第 41～52 行是提示显示与隐藏的触发事件，这里使用了元素的鼠标悬停与鼠标离开事件。下面给出插件中要使用的两个内置对象的定义：

```
/*
/    事件设定类定义
*/
function eventScheduler(){}
eventScheduler.prototype = {
    set : function (func,timeout){
        // set 方法设置了两个参数
        //一个是需要执行的函数，另一个是等待执行的时间周期
        this.timer = setTimeout(func,timeout);
    },
    clear: function(){
        // 清除 set 方法中定义的时间周期
        clearTimeout(this.timer);
    }
}
```

这段代码是我们的事件处理加工对象，主要负责提示隐藏周期设置和当提示显示时，删除隐藏周期。

```
1   /*
2   /    提示对象定义
3   */
4   function Tip(txt){
5       this.content = txt;
6       this.shown = false;
```

```
7    }
8    Tip.prototype = {
9        generate: function(){
10           // generate 方法产生提示对象
11           // 存储在 tip 变量中
12           //并交给插件进行处理
13           return this.tip || (this.tip = $('<span class="colorTip">
             '+this.content+
14                                   '<span class="pointyTipShadow">
                                     </span><span class="pointyTip">
                                     </span></span>'));
15       },
16       show: function(){
17           if(this.shown) return;
18           // 设定插件显示样式并以淡入效果出现
19           this.tip.css('margin-left',-this.tip.outerWidth()/2).fadeIn
             ('fast');
20           this.shown = true;
21       },
22       hide: function(){
23           this.tip.fadeOut();
24           this.shown = false;
25       }
26   }
```

上述代码是提示对象的定义，第 9～15 行利用 HTML 标记<span>组成提示的显示样式，并以 jQuery 对象形式保存交给插件处理。第 16～21 行是提示对象的显示功能。第 22～25 行是提示的隐藏功能。

具体效果如图 15.9 和图 15.10 所示。

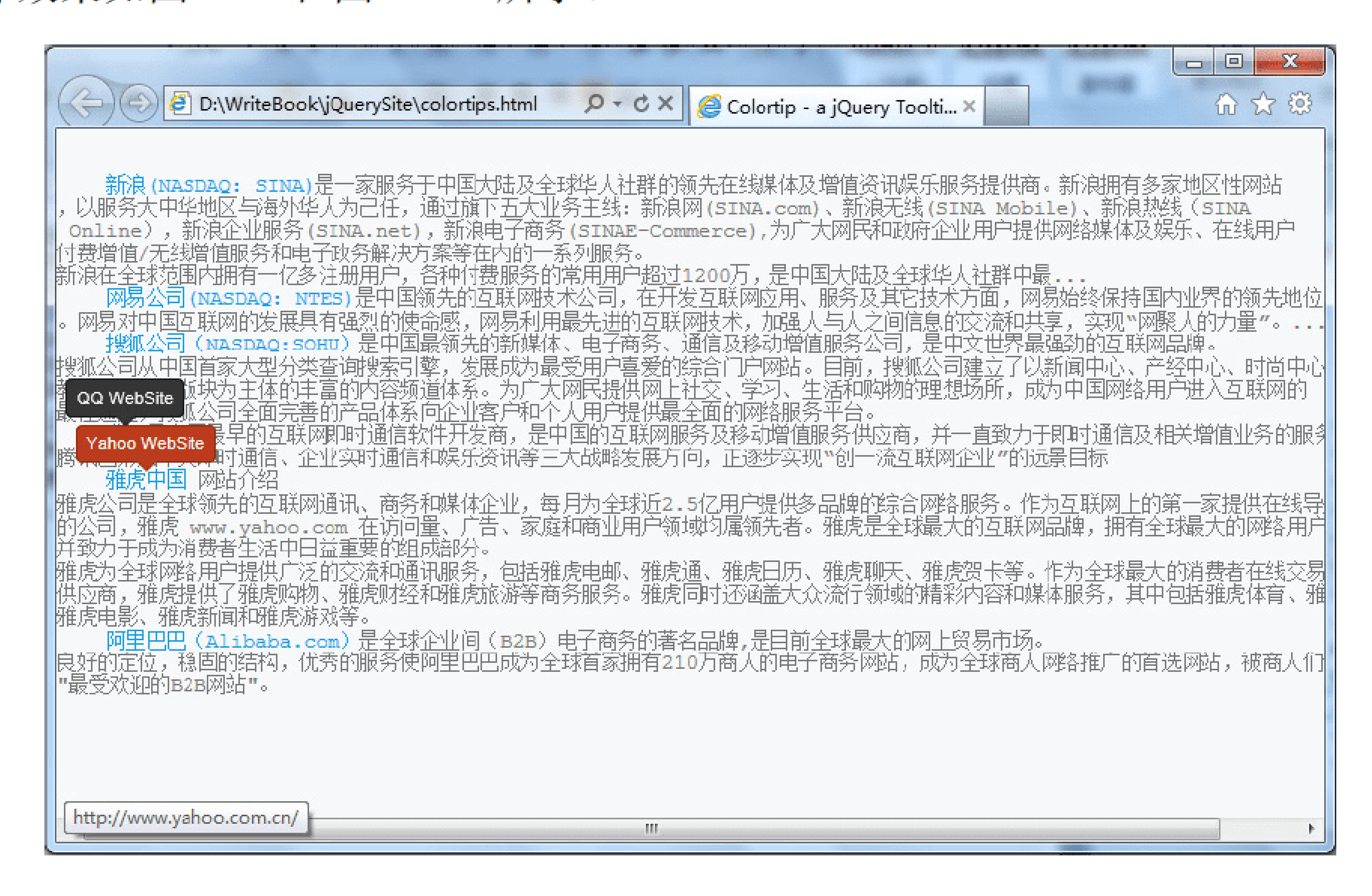


图 15.9　自定义插件效果一

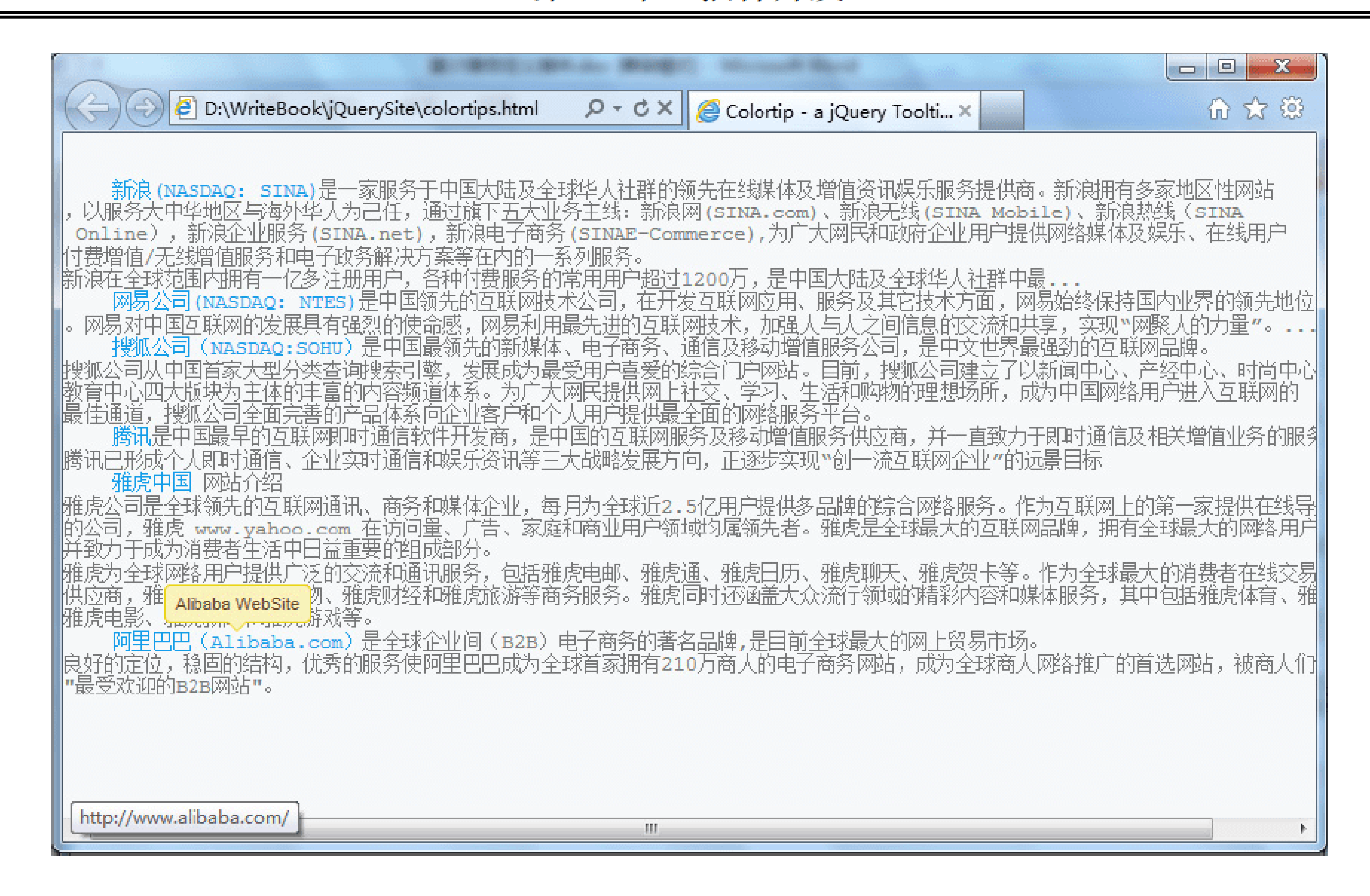


图 15.10　自定义插件效果二

15.3　插件开发规范

前面介绍了 jQuery 插件的相关方法，本节总结一下在开发 jQuery 插件的过程中应该遵守的一些规范。

（1）使用闭包。这个规范在前面介绍插件基本知识的时候已经介绍过，它的优点如下。

- 避免全局依赖。
- 避免第三方破坏。
- 兼容 jQuery 操作符。

（2）关于 this 关键字的使用。

- 不要重复包装 this 关键字。
- 除非你需要返回一个固有的值，否则必须返回 this 引用，以保持 jQuery 方法的链接性。

（3）通过选项对象传递参数给插件，而不要使用零散的参数形式。

（4）不要在插件中出现杂乱的插件命名空间。

（5）必须为插件的方法及事件命名。

（6）利用 jQuery 提供的扩展功能，建议使用$.fn.extend 而不是$.extend。

$.extend 用于扩展自身方法，如$.ajax, $.getJSON 等，$.fn.extend 则用于扩展 jQuery 类，包括方法和对 jQuery 对象的操作。为了保持 jQuery 的完整性，建议使用$.fn.extend 进行插件开发而尽量少使用$.extend。

（7）选择器的使用规范如下。

- 尽量使用 ID 选择器。
- 在样式选择中尽量搭配上标记名。

❑ 避免使用选择器迭代。

15.4　小　　结

本章主要介绍了 jQuery 插件的基础知识，以及如何开发插件。重点内容是如何开发具有一定功能的插件，以及在开发过程中要遵守的插件开发规范。插件开发是本章的难点部分。

15.5　习　　题

【习题】利用本章所讲内容，自定义一个控制元素显示与隐藏的插件，并可更换元素背景色，要求符合插件编写规范。